



SCHOOL OF MANAGEMENT

TECHNISCHE UNIVERSITÄT MÜNCHEN

Interdisciplinary Project

# ETF Portfolio Optimization

Authors:	Eric Armbruster, Ruben Bachmann
Supervisor:	Prof. Dr. Christoph Kaserer
Advisor:	Steffen Windmüller
Submission Date:	15.11.2020



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Background</b>	<b>2</b>
<b>3</b>	<b>Project Requirements</b>	<b>5</b>
3.1	Functional Requirements . . . . .	5
3.1.1	Data Extraction and Management . . . . .	5
3.1.2	Portfolio Optimization and Visualization . . . . .	6
3.2	Nonfunctional Requirements . . . . .	7
3.2.1	User Experience . . . . .	7
3.2.2	Resilience and Security . . . . .	7
3.2.3	Maintainability and Extendability . . . . .	8
<b>4</b>	<b>Implementation</b>	<b>9</b>
4.1	Architectural Overview . . . . .	9
4.2	Scraping Static ETF Data . . . . .	11
4.2.1	Scraping of Justetf Website . . . . .	12
4.2.2	Scraping of Extraetf Website . . . . .	13
4.3	Retrieving of Refinitiv Price Data . . . . .	13
4.4	Creation of the User Interface . . . . .	15
4.4.1	Command-Line Interface . . . . .	15
4.4.2	Graphical User Interface . . . . .	15
4.5	Implementing the Optimizer . . . . .	16
<b>5</b>	<b>System Operation</b>	<b>18</b>
5.1	Installation . . . . .	18
5.1.1	Dependencies Overview . . . . .	18
5.1.2	Windows . . . . .	18
5.1.3	Linux . . . . .	20
5.2	Post Installation and Configuration . . . . .	21
5.3	Command Line Tool Overview . . . . .	22
5.4	Usage Tutorial . . . . .	23
5.4.1	Crawling Data (justetf, extraetf) . . . . .	23

## *Contents*

---

5.4.2	Retrieving Price Data (Refinitiv) . . . . .	23
5.4.3	Exporting and Importing the Database . . . . .	24
5.4.4	Using the Optimizer . . . . .	24
<b>6</b>	<b>Evaluation</b>	<b>26</b>
<b>7</b>	<b>Conclusion</b>	<b>28</b>
	<b>List of Figures</b>	<b>29</b>
	<b>List of Tables</b>	<b>30</b>
	<b>List of Abbreviations</b>	<b>31</b>
	<b>Bibliography</b>	<b>32</b>

# 1 Introduction

Exchange-Traded Funds (ETFs) have been a highly popular investment option in recent years [24] due to their good returns at comparatively low risk. Because of this popularity, many ETFs have been founded, effectively making investment decisions more difficult. In a world with a diverse range of ETFs available, it is natural for investors to want to select ETFs according to certain criteria, such as specific markets, countries or currencies, while maintaining the best possible diversification.

Portfolio optimization is a technique that can help investors make just such decisions. It provides optimization methods that try to predict the future development of investment options such as ETFs by looking at their past performance. As the topic has already been researched for a long time, numerous optimization methods (see [5] for an overview) have been developed. These can be tailored to the specific needs of investors. For instance, some approaches take into account the willingness of investors to take risks.

This interdisciplinary project aims to create an application that allows users to calculate an optimal portfolio consisting of ETFs. The application should also be able to take the previously described investment preferences of the user into account, and provide an easy-to-understand visualization of the optimization results via a Graphical User Interface (GUI).

This project report is structured as follows. Firstly, the mathematical background of portfolio optimization is shortly presented in Chapter 2. Then, functional and nonfunctional requirements of this project are defined in Chapter 3. Afterwards, in Chapter 4, the implementation of the application as well as the fulfillment of the requirements are discussed. Chapter 4 features the installation and setup process as well as the application usage. Then, in Chapter 6, we provide a short evaluation on the optimization performance achieved. Finally, the report is concluded in Chapter 7.

## 2 Background

Before discussing the requirements and implementation of this project, we shortly provide a summary of the most important terms and definitions regarding the optimization methods we used in this project. For our application three types of optimization methods are used, of which two are mean-variance optimizations, and the last one is a maximum sharpe optimization. These optimization methods are derived from the Modern Portfolio Theory (MPT), which was introduced by Markowitz in his 1952 paper “Portfolio Selection.”

The methods discussed in this project are all concerned with finding an efficient portfolio. A portfolio is said to be *efficient* if its expected return is the maximum among all portfolios with the same variance, or if its variance is the minimum among all portfolios with the same expected return [5, p. 142]. The set of all efficient portfolios is called the *efficient frontier*.

Calculating precise values for expected returns and risk is a difficult task, however there are several methods to estimate these values. To estimate expected returns several models may be used of which the following three will be used in our project. Firstly, the most classical method in MPT is to take the mean of the historical price data. Another estimator for expected returns is the exponentially-weighted mean of historical returns [11]. In comparison to the classical mean, this estimator puts higher weight on more recent data [11]. Lastly, the Capital Asset Pricing Model (CAPM) may be used as an estimator of expected returns [11]. The CAPM is equal to the risk-free rate added some risk premium weighted by the asset beta [2].

A typical model for quantifying risk in a portfolio is the covariance matrix. This matrix captures the correlation between individual assets and the volatility of each asset. Positive covariance between assets causes them to move into the same direction and negative covariance to move into inverse directions. Thus, a highly correlated portfolio, i.e. a risky portfolio, is more prone to market volatility. In MPT negative covariance is used to create a diversified portfolio [13].

To predict the risk of a portfolio, the covariance matrix needs to be estimated. For instance, through a sample covariance matrix, a semicovariance matrix [9], or through a more advanced approach that shrinks the covariance matrix. Such a shrunken covariance matrix can be obtained by adapting simpler types of covariance matrices with a shrinkage constant [15, 14].

Mean-variance portfolio optimization is defined by the trade-off between the expected return and the risk of a portfolio [5]. The choice of an optimal portfolio therefore depends on the willingness of the investor to take risks. This problem can be approached in two different ways, either by the maximization of the mean-return, or by the minimization of the risk [5]. These approaches generally result in different efficient portfolios and can be formulated using the following constraints [18].

Maximization of mean-return:

$$\begin{aligned} \max_w \quad & w^T \mu \\ \text{s.t.} \quad & w^T \Sigma w \leq \alpha \\ & 1^T w = 1 \end{aligned} \tag{2.1}$$

Minimization of risk:

$$\begin{aligned} \min_w \quad & w^T \Sigma w \\ \text{s.t.} \quad & w^T \mu \geq \beta \\ & 1^T w = 1 \end{aligned} \tag{2.2}$$

In both cases  $w$  are the asset weights of the portfolio,  $w^T \mu$  is the expected return of the portfolio,  $w^T \Sigma w$  is the risk of the portfolio,  $\alpha$  is the maximum risk and  $\beta$  is the minimum return. These last two parameters,  $\alpha$  and  $\beta$ , are tuning parameters that may be adjusted by investors according to their willingness to take risk and their target return. By solving either one of these quadratic problems an efficient portfolio can be acquired.

As mentioned previously, we also offer a maximum sharpe optimization. This optimization method results in an optimal risky portfolio by maximizing the sharpe ratio [5]. The sharpe ratio is a measurement for the reward at a level of risk and uses the risk-free rate as a baseline [28, 29]. If the return of the optimal portfolio is equal or lower than the return of the risk-free asset, the investor should invest in the risk-free asset instead. By using this method a more diversified portfolio can be obtained, as with increased diversification, risk decreases and the sharpe ratio becomes larger. This maximization is described by the following constraints [18].

$$\begin{aligned} \max_w \quad & \frac{w^T \mu - r}{\sqrt{w^T \Sigma w}} \\ \text{s.t.} \quad & 1^T w = 1 \\ & w \geq 0 \end{aligned} \tag{2.3}$$

As in the formulas of the mean-variance optimization,  $w$  are the asset weights of the portfolio,  $w^T \mu$  is the expected return of the portfolio,  $w^T \Sigma w$  is the risk of the portfolio. The variable  $r$  stands for the risk-free rate.

Depending on the requirements of the investor each of these optimization methods offer different advantages and disadvantages. However, they all calculate efficient portfolios at different points on the efficient frontier, with the riskier portfolios offering higher returns. By offering these three optimization methods we give the investor options to choose preferences in terms of risk and potential return.

## 3 Project Requirements

At the beginning of this project we define multiple functional and nonfunctional requirements for our application. These requirements must be fulfilled to successfully complete this project and will be described further in the upcoming sections.

### 3.1 Functional Requirements

The following subsections describe the functional requirements of the `etfoptimizer` tool. The main objective is to create an application that enables the user to initially select a set of ETFs and then allows the user to determine an optimized ETF portfolio based on historical price data.

#### 3.1.1 Data Extraction and Management

Initially, this requires the application to extract ETF data from various data sources. The data consists of two parts. Of which the first is static data, such as names, International Securities Identification Numbers (ISINs) and categories of ETFs. Whereas the second is dynamic data, such as the prices of the respective ETFs. In order to fulfill this requirement, the user should be able to extract and manage both types of data with a Command-Line Interface (CLI) tool.

First, the user should be able to extract the static data into a previously set up database. The use of multiple data sources is required here, since not all the required data is offered by a single data source. Additionally, the CLI tool should offer the user the ability to delete the static data from the database, which is useful in the rare case that the static data changes and needs to be extracted again.

Furthermore, the CLI tool should also enable the user to extract the dynamic data into this database. For the dynamic data one of two different possible extraction methods, which either gather the data from an Excel sheet or from an online database, should be offered to the user. The chosen method should support the addition of new values at a later point in time. With the static and the dynamic data the user should then be able to calculate an optimal portfolio via the GUI.



Finally, the user needs the ability to export and import the database. This is helpful in case the data needs to be transferred between systems without re-extracting. Combined, these functionalities enable the user to easily extract and manage the required data.

#### 3.1.2 Portfolio Optimization and Visualization

As mentioned previously, the main goal of this project is to provide an application, which is able to calculate and visualize an optimal portfolio consisting of ETFs. In order to fulfill this objective, a GUI with the following functionalities is required.

1. Selecting ETFs that should be considered
2. Choosing from various models for risk and return and from various optimization methods
3. Setting optimization parameters
4. Visualizing the optimization results

The first of these requirements enables the user to select the ETFs that are considered for the optimization. For this requirement to be fulfilled, the User Interface (UI) should provide multiple dropdown lists that filter the considered ETFs according to selected categories (e.g. asset class, country of an ETF). Furthermore, the user should be able to select ETFs that are considered in addition to the ones selected with the categories. This ensures that the user has as much control of the considered ETFs as possible.

Another required feature is the possibility to select a risk and return model and an optimization method. As before, this may best be implemented via a dropdown list. By selecting different models for risk and return and trying different optimization methods, the user may compare the results and gather more information about a potentially optimal portfolio.

To fulfill the third requirement, the UI must provide the appropriate input fields through which the user can set the optimization parameters. Additionally, the user should be able to decide if a historic performance of the optimal portfolio should be calculated and plotted. After setting these values the user can click a button for starting the optimization. In case the user entered incorrect values, or did not select any ETFs for optimization, a warning should be shown, informing the user about the incorrect values. This also ties in with the nonfunctional requirements of user experience and resilience, which are explained in more detail below.

For the final requirement the results of the optimization should be shown at the bottom of the UI. These results should include the expected annual return, the annual volatility as well as the sharpe ratio of the calculated optimal portfolio. Below these

values multiple tabs should allow the user to switch between graphs. These graphs include a table showing the weight as well as the amount to be bought of each ETF, a pie chart visualizing the same data, a comparison of the ETFs against the efficient frontier and a historic performance of the portfolio. The historical performance graph is intended to show the return that an investor would have generated over the last three years if he or she had invested in the optimized portfolio three years ago and held it until now.

## 3.2 Nonfunctional Requirements

The following requirements describe the qualities beyond the functionalities our application has to fulfill.

### 3.2.1 User Experience

One of the most important nonfunctional requirements for the application is to provide a good user experience. Therefore, the runtime of the optimization should be kept as low as possible, since having to wait for extended periods of time would greatly impact the user experience negatively. This is especially important in our case because portfolio optimization, if done carelessly, can take a lot of time.

Additionally, the user should be able to easily understand and use the application, which is why the UI has to be designed with care. This includes a user-friendly design, as well as tooltips, which can help users better understand the input values for the optimizer. Overall, the goal should be to make the application as user-friendly as possible.

### 3.2.2 Resilience and Security

Furthermore, invalid user input must be handled with special care to avoid any kind of crashes and to enable the application to recover from an exceptional state. Thus, helpful error message should be shown to the user, explaining why an input was invalid. This also indirectly affects the user experience, since a crash is more likely to frustrate the user than a simple error message.

In addition to that, the goal is to design and implement a secure web application that is not vulnerable to any kinds of common vulnerabilities such as those listed in the OWASP Top Ten [20]. For instance, an attacker should not be able to alter or retrieve arbitrary contents of the database by using SQL injection attacks.

### **3.2.3 Maintainability and Extendability**

The final nonfunctional requirement for the application is low maintenance. The only maintenance required should be updating the dependencies for security reasons and making code changes if required by new versions of dependencies.

Furthermore, the application should be easy to extend by future developers, which can be achieved by properly documenting the code and designing code interfaces with extensibility in mind.

## 4 Implementation

This chapter describes the decisions we made during the design and implementation phase of the *EtfOptimizer* tool. It begins by providing a short architectural overview in Section 4.1, continues with an explanation on how data was scraped from various sources in Section 4.2. Afterwards, we discuss retrieving dynamic price data from *Refinitiv* in Section 4.3. In Section 4.4 we discuss the implementation of the optimization UI and the implementation of the CLI that is intended for administrating the application. The chapter is finished in Section 4.5 with an explanation on the implementation of portfolio optimization. Section 4.5.

### 4.1 Architectural Overview

For the *EtfOptimizer* tool we decided its best to use a classical three-tier architecture, which is a type of client-server architecture. In such an architecture, there is a clear distinction between three tiers. The tiers are Presentation, Logic and Data which are responsible for functions of the same name within the application. Where possible, we separated these concerns in our code. A purely classical three-tier architecture, however, proved insufficient for the use cases we described in Chapter 3. Thus, we decided to add a second presentation tier in the form of the CLI besides the GUI.

Figure 4.1 shows the most important components of the *EtfOptimizer* architecture. The two presentation tiers are named there *EtfOptimizer* Frontend (GUI) and *EtfOptimizer* Command-Line Client (CLI). Next, the Chrome Driver component is used for scraping data from `justetf.com`. Furthermore, the Optimizer is an external solver performing linear programming, quadratic programming, mixed integer programming and more. Finally, there is a database component for storing scraped data and retrieving it for optimizations.

There are several possible deployment scenarios for *EtfOptimizer* of which two are modelled as UML deployment diagrams here. In Figure 4.2 we show the most basic scenario, in which a data server is hosting the web application and is responsible for retrieving all the data and a user that only needs a web browser to access the optimizer via the GUI.

In Figure 4.3 a different deployment scenario is shown. In this scenario two peers both start up their own application server locally. On a side note, the term peer is not

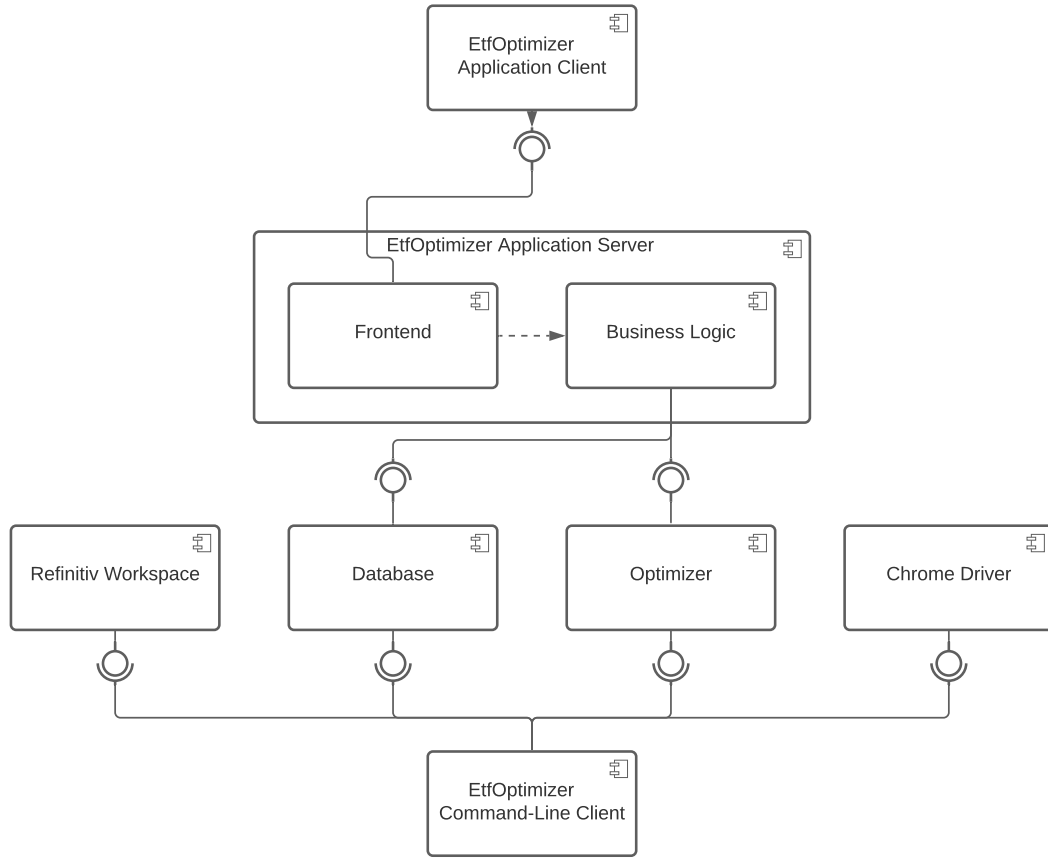


Figure 4.1: UML component diagram showing the architecture of *Etfoptimizer*

really fitting here, but for the lack of other terminology we use it here. The fat peer has access to Refinitiv Workspace and the Chrome Driver. Thus, this peer is responsible for retrieving the data and providing the data to the other peer(s) whenever updated data is required. Transferring the data must be done manually via the import export functionality of *Etfoptimizer*.

Finally, we want to emphasize that the architecture allows some flexibility, as it has two optional components, which are Refinitiv Workspace and the Chrome Driver. Without Refinitiv it is possible to run on Linux and the Chrome Driver may be removed when not needing to scrape data. Also, other components may be switched to different implementations with relative ease due to the libraries we used. For instance, the database implementation and the optimizer implementation may be changed to other

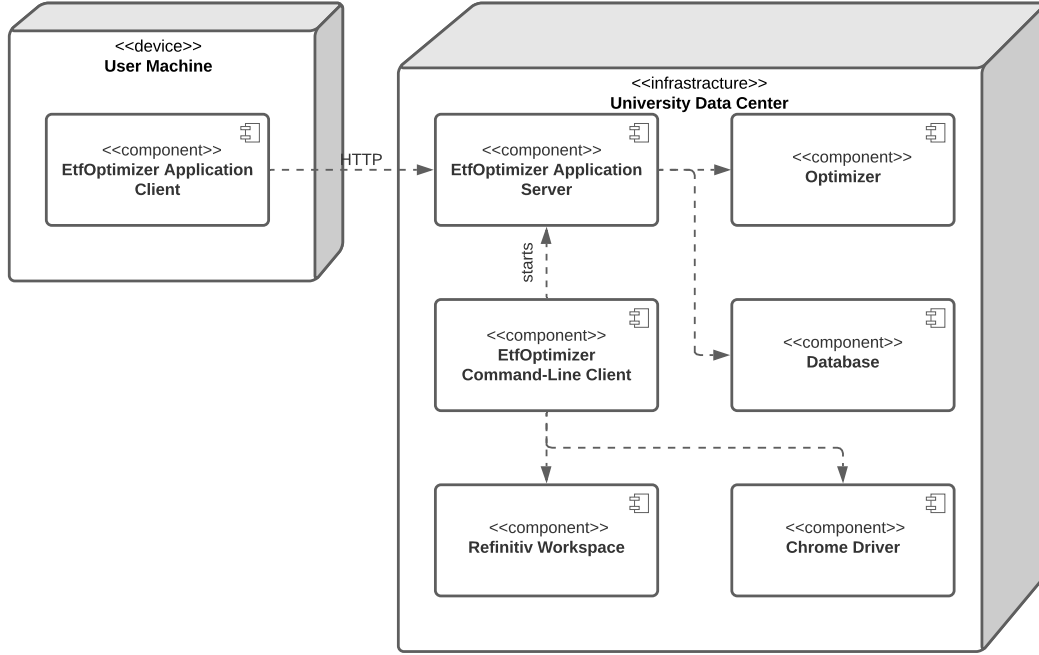


Figure 4.2: *EtfOptimizer* server deployment

implementations that are supported by SQLAlchemy [31] and CVXPY [6]. We decided for PostgreSQL [22] as database, as it is a relational database used throughout the industry since many years and there is no need for any specialized database. We chose Gurobi [12] as optimizer, as it is industry-wide used, fast and free of charge for academic purposes.

## 4.2 Scraping Static ETF Data

*Web scraping* refers to automating the process of retrieving, extracting and analyzing data from websites typically via the HTTP protocol. A *crawler* or *spider* is a bot scraping a defined set of URLs and following any URLs found to scrape these sites as well. The process starts by downloading a site, then parsing the HTML returned and extracting information and URLs from it.

In the following two subsections Section 4.2.1 and Section 4.2.2, we will shortly provide how we built the two crawlers for `extraetf.com` and `justetf.com`.

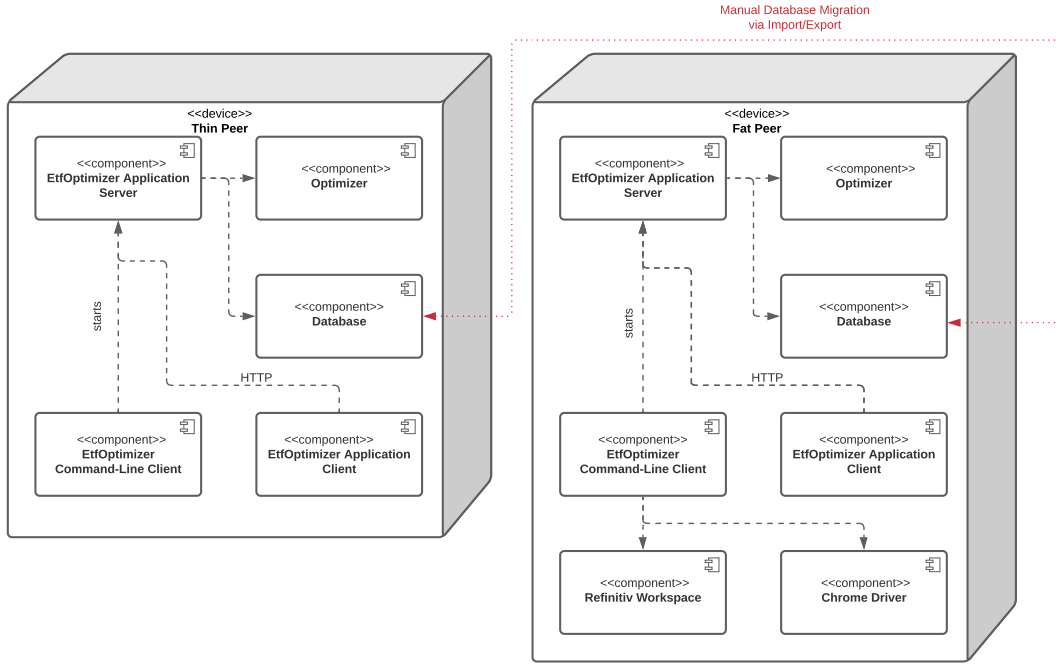


Figure 4.3: *EtfOptimizer* peer deployment

### 4.2.1 Scraping of Justetf Website

To scrape justetf.com, we decided to use Scrapy, a Python module for high-level crawling that provides a framework for building crawlers cleanly and easily [26]. However, we quickly realized this to be insufficient for crawling justetf.com in particular, as the website must execute Javascript when clicking on the next page button in the ETF table overview and libraries like scrapy only support static HTML code. Thus, an automated browser accessing these pages and running the Javascript is additionally required. We are using Selenium with Python [27] for automating Chrome [3] with the Chrome Driver [4].

In Figure 4.4 we show the main loop of the justetf.com spider. After retrieving the page through the driver, a Selector is used for extracting the links to the ETF detail pages. For each link, a Request is generated, i.e. the page at the link is downloaded and parsed according to the parse\_item() method. The XPath language is used for navigating the tree like HTML source code to get to specific elements (moving from the root HTML to the inner node or leaf that is of interest) [33]. Finally, the next page button is searched. If the button is disabled, the crawler stops. Otherwise, the next

```
pagenum = 1
while True:
    r = Selector(text=self.driver.page_source)
    for link in r.xpath('//tbody/tr[@role="row"]'
                        '/td/a[@class="link"]/@href').getall():
        yield Request(link, callback=self.parse_item)
    pagenum += 1
    next_page = \
        self.driver.find_element_by_xpath('//a[@id="etfsTable_next"]')
    disabled = next_page.get_attribute('class').find('disabled')
    if not next_page.is_enabled() or not next_page.is_displayed() \
        or disabled != -1:
        break
    self.driver.execute_script("arguments[0].click();", next_page)
```

Figure 4.4: Main loop extract of justetf.com crawler

page is loaded by clicking on the button. The `parse_item()` method is not presented here as it mainly performs selections of specific information via XPath, which is similar to the previously shown XPath. Afterwards, the extracted data is put into a suitable format and stored into the database.

#### 4.2.2 Scraping of Extraetf Website

We planned on scraping the `extraetf.com` website with Scrapy as well, however, this is not required, as we found an unofficial and undocumented Application Programming Interface (API) that is used by the site internally and can be used by us as well. This means HTTP GET requests are sent to the API endpoints, which return the data already in JSON format, which is easily parsable for machines. The steps are then the mostly same as for `justetf.com`, i.e. the data is formatted a bit and written into the database. There is one additional step, as we make the assumption that data from `justetf.com` could already be in the database. As a consequence, if an ETF is listed on both sites, we need to choose which values to keep from which site.

### 4.3 Retrieving of Refinitiv Price Data

For the retrieval of the price data we decided to use the Eikon API [8], which offers multiple ways to extract the necessary data. The ability to retrieve time series data



```
for i in range(0, len(isins)):
    ric = ek.get_data(isins[i], ['TR.LipperRICCode'])[0].values[0][1]
    if isinstance(ric, str):
        today = (date.today()).strftime('%Y-%m-%d')
        data = ek.get_timeseries(ric, fields=['TIMESTAMP', 'VALUE'],
                                start_date=start_date, end_date=today, interval='daily')
        for j in range(0, len(data.values)):
            isin = isins[i]
            datapoint_date = data.axes[0][j].date()
            price = data.values[j][0]
            __write_history_value(isin, datapoint_date, price, session)
        skipped_isins.remove(isins[i])
```

Figure 4.5: Loop requesting time series data via Eikon API (extract)

was of particular use, since the price data of the past several years is required for each ETF. One of the problems with the retrieval of the time series data was, that not all ETFs returned data with a single type of request. This is handled by using two types of requests. Of which the first requests data for all ETFs, as shown in Figure 4.5. The second then only requests data for the ETFs that did not return any data during the first run. For both request types we implemented this as a loop over the ETFs. This loop immediately writes any available data to the database.

In order to support updating the price data without having to retrieve the old data again, the date most recently written to database is looked up at the beginning of the process. The start date of the extraction process is then set to this date. In case no data has been written to the database yet, the start date is set to the first of January 1990. Additionally, the end date is set to the current date. This way only the new data needs to be retrieved.

Another way of retrieving the price data would be to extract the data from an Excel sheet. This could be done by first extracting all ISINs into a separate Excel sheet, and then using these in combination with the *Refinitiv Excel add-in* to extract the necessary data. After that, the data would have to be transferred from the Excel sheet to the database. Even though we initially implemented this method, it proved to be error-prone, superfluous, as the data may as well be retrieved via the API, and labor-intensive for the application user, as many manual steps would be required. This is why we ultimately decided to abandon this method.

## 4.4 Creation of the User Interface

The following sections describe the implementation of the user interface. The first section focuses on the implementation of the command-line interface and the second section describes the implementation of the graphical user interface.

### 4.4.1 Command-Line Interface

The command-line interface is intended mainly as a frontend for all administrative tasks, i.e. for retrieving static and dynamic site data and importing and exporting the database, but it is also used for starting the GUI. We used the Click [19] Python module for implementing this, as this library elegantly hides the complexity of parsing arguments from command-line and provides an easy-to-use structure.

### 4.4.2 Graphical User Interface

For our graphical user interface we decided to use Dash [7], a framework for building interactive UIs with Python. There are two main reasons we decided to use Dash. Firstly, it offers all the functionality we need for our UI, like dropdown lists and input fields, which are relatively easy-to-understand and use. Secondly, it supports different styles, which makes it possible to not only create a functional but also a good-looking UI, effectively improving the user experience.

Dash allowed us to design the code of our GUI in a modular way. This means, that the elements of the UI, like dropdown lists or input fields, can be created independently with simple function calls. These elements can then be used in different combinations to create the UI. An example for such an element is given in Figure 4.6, which shows the code for the creation of a dropdown list. This design should make it relatively easy to extend the UI in the future if needed. In addition, Dash enables easy retrieval of inputs from different UI elements and allows executing code depending on inputs. This makes it easy to create a UI with the required functionality.

For the visualization of the optimization results we decided to use Plotly [21]. Plotly offers a wide range of functions for easily creating and styling graphs, which helped us with building our own graphs. We mostly required line graphs for showing price data curves but also utilized some scatter graphs for the efficient frontier and a pie chart for visualizing the weights from the allocation result. Also, Plotly offers by default graph interaction features, such as zooming and range selection, which can help with the understandability and interpretability of the optimization result.

```
def create_dropdown(dropdown_id, dropdown_data,
                    width, dropdown_multiple, default_value=None):
    dropdown = html.Div([
        html.Label(dropdown_id + ':'),
        dcc.Dropdown(
            id=dropdown_id + ' Dropdown',
            options=[{'value': data_id, 'label': data_name}
                     for data_id, data_name in
                     sorted(dropdown_data, key=lambda x: x[1])],
            placeholder=dropdown_id,
            searchable=True,
            clearable=True,
            className="dash-bootstrap",
            multi=dropdown_multiple,
            value=default_value)])
    return dropdown
```

Figure 4.6: Function for creating a dropdown list (extract)

## 4.5 Implementing the Optimizer

Implementing a custom optimizer is by no means a realistic task for a student project, as the task itself would be enormous and is also not needed as numerous proprietary (e.g. Gurobi [12]) and open source solvers are well-established. Homegrown solutions do not stand any chances to easily reach the kind of programmatic optimization level that is typical of these products and required to solve large problems.

So what we mean by implementing the optimizer, is essentially combining all the different parts that are required for performing the optimization. The optimization problems described mathematically in Chapter 2 are handled for us by PyPortfolioOpt [17], a portfolio optimization module written in Python. Internally, PyPortfolioOpt uses CVXPY [6], a modelling language for convex optimization problems. This language abstracts away the details of any specific solver implementation, meaning numerous solvers may be used with it and in turn also with our application.

As mentioned previously the main task here is combining all the different parts, which is still complex due to the large number of libraries involved, all expecting different data formats in their method calls. We summarize the main steps that are required after pressing the optimize button in the UI as follows:

1. Validate all input fields. Convert them into appropriate data types.
2. Retrieve the ISINs matching the category selection and the price data of the last three years from the database. Validate that the selection results in ISINs that are available in the database and that there is price data available.
3. Prepare the optimizer using the chosen models for risk and return.
4. Run the chosen optimization.
5. Perform the allocation for the investment amount using the chosen algorithm. PyPortfolioOpt offers two allocation algorithms, one that is greedy and fast and another one that treats discrete allocation as an integer programming problem, meaning optimization is done to find an allocation that is as close as possible to the actual weights stemming from the optimization.
6. Convert the allocation results into the appropriate data formats for visualization with Plotly.
7. Create the visualizations, i.e. the efficient frontier graph, the pie allocation, the allocation data table and the historical performance graph (which requires running another optimization on the price data from three to six years ago)

# 5 System Operation

## 5.1 Installation

The installation of *EtfOptimizer* is nontrivial due to the large number of software libraries and dependencies used. Please make sure to have at least 8 GB of disk space available on the installation machine.

### 5.1.1 Dependencies Overview

- Python 3 ( $\geq 3.8$ )
- C++ Build Tools (Windows only, pre-installed on Linux)
- PostgreSQL database
- The Gurobi solver (free for Academic Purposes)
- Optional: Chrome, ChromeDriver (for ETF web scraping)
- Optional: Refinitiv Workspace (for ETF price data retrieval, only available for Windows).
- Several Python packages

### 5.1.2 Windows

#### Overview

1. Install Python 3 [23]
2. Install Build Tools for C++ [1] <sup>1</sup>
3. Install the Gurobi solver [12] (free for Academic Purposes).
4. Install the PostgreSQL database [22].

---

<sup>1</sup>Download the **Visual Studio Installer** [32], however installing Visual Studio is not required.

5. Install *EtfOptimizer* repository [10] and its Python dependencies.
6. Optional: Install Chrome [3] alongside with a compatible version of its web driver, the ChromeDriver [4].
7. Optional: Install Refinitiv Workspace [25].

### Detailed Instructions

In the rest of this section we provide detailed instructions and tips on how the dependencies need to be installed.

**Python Installation:** Download the latest Python 3 installer and follow the installation guide [23]. Make sure to select *Add Python to PATH* in the installer. We strongly recommend at least Python 3.8 or later, as we have only tested with this version. Make sure the installation process was successful. For this, press *Win + R*, type *powershell* and hit Enter. Type *python -V* into the terminal and hit enter again. In case no Python version is shown, but a message like *unknown command*, that means something went wrong. Most likely the Python installation could not be found. Follow *this guide* to set the PATH environment variable to the Python installation location. Then check again in Powershell that the version is shown.

**Build Tools for C++** The optimizer relies on build tools for C++. These can be installed on Windows best by downloading the Visual Studio installer[32]. Then follow this guide to install the required packages [1]. For completeness, we also list them here: C++ build tools (MSVC, Windows 10 SDK, C++ CMake, Testing tools core features - Build Tools).

**Gurobi** Please install Gurobi as described on the Gurobi website [12]. Make sure to get an Academic License.

**Chrome and ChromeDriver** This should be straightforward. Download and install Chrome [3] and ChromeDriver [4]. Make sure to pick compatible versions. These dependencies are only required for retrieving data from *justetf.com*. Uninstalling them without causing problems should be possible once retrieving data from *justetf.com* is no longer required.

**PostgreSQL** Please follow the installer from the official download site [22]. **Important:** Please ensure the "bin" folder of the PostgreSQL installation has been added to the PATH environment variable, otherwise database imports/exports via command-line will not work.

1. Open *Environment Variables* through the Windows search menu.
2. In the section *System Variables* find the variable PATH.
3. Edit the variable and add the path to the *bin* folder of the Postgres installation location. By default, PostgreSQL should be installed to *C:\Program Files\PostgreSQL\*

Please verify that everything went well by opening a Windows Terminal and running `pg_dump` and `psql`. The installation was done correctly if these commands do not return an unknown command error.

**Refinitiv Workspace** Please follow the official instructions for installing Refinitiv Workspace [25].

**Etfoptimizer** To install *EtfOptimizer* first download the latest version from Gitlab. Unzip the downloaded file and **go into the *etfoptimizer* directory**. Then run the following commands in a Windows Terminal (e.g. Powershell):

```
$ python3 -m venv venv
$ venv\Scripts\activate.bat
$ pip install --editable=.
```

If everything went well the message "Successfully installed EtfOptimizer-1.0.0" should be shown. This has also created a virtual environment to ensure that there will be no version clashes with other Python projects installed locally. **Please make sure to always run `venv\Scripts\activate.bat` in the respective folder before running any etfopt commands, i.e. each time a new terminal must be opened. Otherwise, etfopt will be an unknown command.**

### 5.1.3 Linux

The following steps are provided for the Arch Linux distribution, but should be similar for other distributions. Please note, that on Linux retrieving price data is not possible, as Refinitiv Workspace cannot be installed. Instead, it must be imported via the database import functionality, see Section 5.4.3. The dependencies can be installed via command line:

```
$ pacman -S python chromium postgresql
```

Gurobi can be installed from the Arch User Repository (AUR) or manually. Please refer to the Gurobi site for manual installation[12]. The installation and correct licensing can be verified by running "gurobi.sh" from terminal. To start the PostgreSQL database and to permanently enable it on for each startup run:

```
$ sudo systemctl start postgresql.service
$ sudo systemctl enable postgresql.service
```

The chrome driver should not be needed on Arch Linux. On other Linux distributions it might need to be installed via the package manager. Also, the build tools which are needed on Windows should be shipped per default on all Linux distributions (gcc, cmake, etc).

Finally, finish the installation by installing the required Python packages. For this, download the *EtfOptimizer* Repository from Gitlab [10]. Then run the following commands.

```
$ python3 -m venv venv
$ source venv/bin/activate
$ pip install --editable=.
```

If everything went well the message "Successfully installed EtfOptimizer-1.0.0" should be shown. This has also created a virtual environment to ensure that there will be no version clashes with other Python projects installed locally. **Please make sure to always run `source venv/bin/activate` in the respective folder before running any etfopt commands, i.e. each time a new terminal must be opened. Otherwise, etfopt will be an unknown command.**

## 5.2 Post Installation and Configuration

This section describes the steps that must be performed after the installation and describes the configuration of *EtfOptimizer*. The configuration file `EtfOptimizer.ini` resides relative to the home folder of the currently logged-in user. On Linux the default path should be `~/.config/EtfOptimizer` and on Windows `C:\Users\<windows user>\AppData\EtfOptimizer\Config`.



Key	Description	Default
dialect	The SQL dialect to use. Depends on the database used [31]	postgres
driver	The driver to connect to the db [30]	psycopg2
username	The db username	*
password	The password for the db user	*
host	The hostname of the db. Either an IP address or a domain name. Use <i>localhost</i> for the current device	localhost
port	The port to connect to the database	5432
database	The name of the database that is used to store the data	etf_optimization
cutoff	The default cutoff. ETFs smaller than cutoff are not considered in the portfolio	0.00001
rounding	The number of trailing decimals to keep when rounding	5
risk_free_rate	The default risk free rate to be used	0.02
total_portfolio_value	The default total portfolio value to be assumed when calculating shares to buy	0.02
app_key	The Refinitiv API key to be used for retrieving price data	*

Table 5.1: Configuration Reference. Values marked with \* must be set after first run of the CLI

### 5.3 Command Line Tool Overview

This section contains a description about each command available for `etfopt`, the command-line tool of this optimizer. Each of the following commands must be prepended with `etfopt`, i.e. in order to use the help command `-help` type `etfopt -help`. The same content is also presented as aforementioned help message.

It is also possible to display the usage of each command. This shows the parameters and options accepted by each command. To show the usage of a command type, `etfopt <command> -help`.

- `-help` Display an overview of available commands and a short description of each command
- `crawl-justetf` Runs a crawler for retrieving data from justetf.com
- `drop-static-data` Deletes tables holding static ETF data
- `export-db` Exports the etf database into a file
- `extract-isins` Extracts all ISINs from db to a csv file
- `import-db` Imports the etf database from a file
- `import-history` Retrieves historic etf data from Refinitiv
- `start-gui` Starts the graphical user interface

## 5.4 Usage Tutorial

In the next sections, usage tutorials are provided for various use cases of *EtfOptimizer*.

### 5.4.1 Crawling Data (justetf, extraetf)

*EtfOptimizer* can retrieve static ETF data from `extraetf.com` and `justetf.com`. If data has already been retrieved using this tool make sure to delete it first via `drop-static-data`. A manual backup of the database is **strongly encouraged**, e.g. via the `export-db` functionality (see ??), as these crawlers are built on an undocumented API, which means they will break eventually. **Please make sure to crawl justetf.com first, as this is expected, if ETFs data is retrieved from both sources.**

```
$ etfopt export-db
$ etfopt drop-static-data
$ etfopt crawl-justetf
$ etfopt crawl-extraetf
```

### 5.4.2 Retrieving Price Data (Refinitiv)

The dynamic ETF data (price data) can be retrieved with the `import-history` command. This automatically retrieves the data from the Refinitiv servers and writes it to the database. By running this command at a later point in time, only the new data will be retrieved.

### 5.4.3 Exporting and Importing the Database

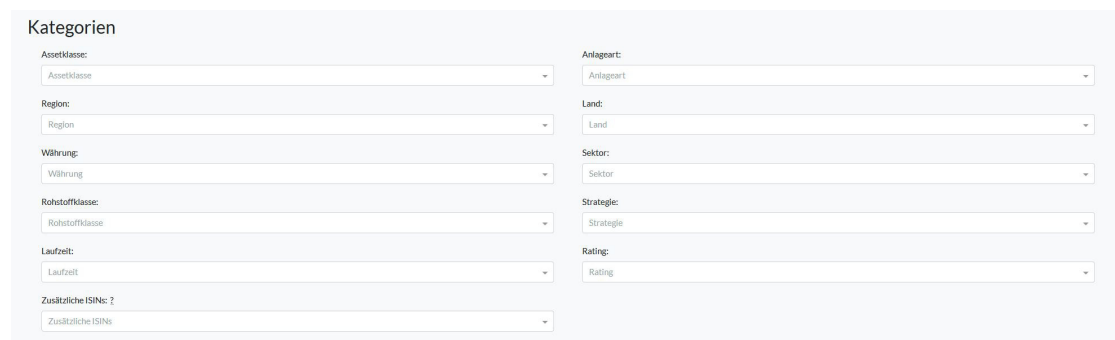
This functionality can be used for creating a backup of the database or for transferring the database from one device to another. To do the latter, run the following commands:

1. On the first device: run `etfopt export-db` to export the database into *backup.sql*
2. Transfer the *backup.sql* file from one device to another, e.g. via a USB drive
3. On the second device: run `etfopt import-db` to import the database from file and make sure the *backup.sql* file is in the project directory of the *EtfOptimizer* (the same directory as on the other machine). Alternatively, a file path may be specified via the `-f` option.

Alternatively, this can also be done manually via many database tools, e.g. *pgAdmin* or *DBeaver*. The functionality in these tools is typically termed backup (dump) and restore.

### 5.4.4 Using the Optimizer

To start the GUI for the optimizer run `etfopt start-gui`. Open the browser and visit <http://127.0.0.1:8050/> to use the optimizer.



The screenshot shows the 'Kategorien' (Categories) section of the EtfOptimizer GUI. It contains two columns of dropdown menus. The left column includes: 'Assetklasse' (Asset Class), 'Region', 'Währung' (Currency), 'Rohstoffklasse' (Commodity Class), 'Laufzeit' (Maturity), and 'Zusätzliche ISINs?' (Additional ISINs?). The right column includes: 'Anlageart' (Investment Type), 'Land' (Country), 'Sektor' (Sector), 'Strategie' (Strategy), and 'Rating'. Each dropdown menu has a small downward arrow icon on the right side of the selection box.

Figure 5.1: Dropdowns for the selection of the ETFs that should be considered

The user first needs to select which ETFs should be considered for the optimization. This selection can be done with the dropdown lists at the top of the GUI, which are shown in Figure 5.1. Every ETF that fulfills at least one of the selected criteria will be considered. The last dropdown can be used to select additional ETFs that should also be considered regardless of the selected criteria.

In the next step the user needs to select the parameters that should be used for the optimization. These include the optimization method and the investment amount.

## Optimierung

Methode:

Mittelwert/Varianz

Investitionsbetrag (€): ? 100000 ✓

Risikofreier Zinssatz: ? 0.02 ✓

Cutoff: ? 0.00001 ✓

☐ Historische Performance berechnen ?

Optimiere

Figure 5.2: Dropdowns and input fields for the selection of the optimization parameters

The optimization method can once again be selected with a dropdown, whereas the other values can be entered into the input fields below, as shown in Figure 5.2. In addition, the user can select with a checkbox whether a historical performance should be calculated. After setting these parameters the user needs to click the button below the checkbox and wait for the results.

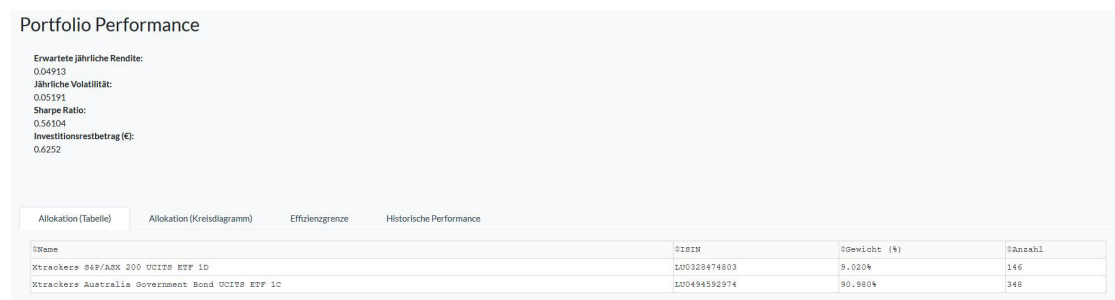


Figure 5.3: Visualization of the optimization results

Figure 5.3 shows the visualization of the optimization results. The values at the top specify parameters like the expected annual return, the yearly volatility, the sharpe ratio and the amount of money that is left over after buying the ETFs according to the optimization results. Below that, the user can switch between different visualizations. These visualizations are a table as well as a pie chart of the allocation, the efficient frontier and optionally a historic performance.

## 6 Evaluation

To evaluate the optimization, we compare the performance of the max sharpe optimization against the MSCI World. The MSCI World index consists of numerous highly valuable stocks and is a strong indicator for the overall market performance, and therefore is also suited as a benchmark.

For this comparison we calculated the historic performance of an optimized portfolio over the past seven years, using price data of the past ten years and a start budget of 100.000 Euro. The optimized portfolio consists only of US assets, and was optimized using the maximum sharpe optimization. To obtain more precise results we decided to recalculate the weights of the assets after each year, which enables the optimization to take more recent price developments into account. This approach also allows us to consider more ETFs for the optimization in the later years, since some of them do not have any data for the first few years.

For each of those yearly steps, the price data of the past three years is considered. Wins and losses of the previous years are carried over, to allow for an accumulation of value over multiple years. After performing these optimizations, we plotted both the MSCI World and our optimized portfolio, which can be seen in Figure 6.1.

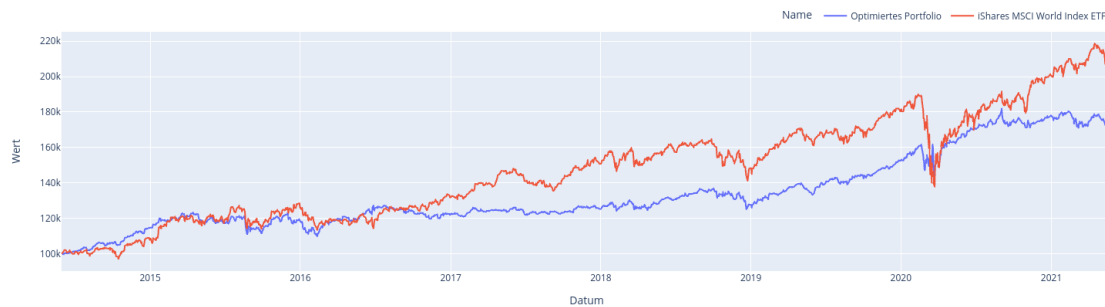


Figure 6.1: MSCI World against max sharpe optimized portfolio

The figure displays the MSCI World in orange, and our optimized portfolio in blue. In the first two and a half years our portfolio performs approximately equal and in some cases even slightly better than the MSCI World. Between 2017 and 2019 MSCI World clearly outperforms our portfolio, resulting in an overall lower portfolio value in 2019. With regard to the relative increase, we can observe that our portfolio performs

similarly from 2019 until mid 2020. During the corona crisis our portfolio proves to be of smaller volatility, as it less susceptible to extreme market behavior, and therefore manages to shortly outperform MSCI World. In the last few months MSCI World again outperforms the optimized portfolio.

Although our optimized portfolio manages to perform about as good as MSCI World for long periods of time, there also are time periods during which it underperforms. A possible explanation for this behavior is, that the risk as well as the expected return necessarily are estimations instead of precise values. Since those are input values for the maximum sharpe optimization, the result of the optimization can be significantly off. Additionally, the portfolios created by the maximum sharpe optimization have a certain level of risk. Therefore, high returns are not always guaranteed. Despite that, our optimized portfolio generates a return of 80% over seven years, which averages out to a yearly return of about 9%. Although the final value of our portfolio is lower than MSCI World, this can be considered an acceptable result.

## 7 Conclusion

In this interdisciplinary project we took on the task to build an ETF portfolio optimizer that relies on data from various sources. To reach this goal, we began by implementing crawlers for retrieving static data from `justetf.com` and `extraetf.com`. Next, we retrieved the dynamic price data from Refinitiv. Afterwards, we finished the implementation by building a UI optimization tool that enables analysts to input their investment preferences and to make an informed decision.

Through this implementation, we have met our functional requirements of data retrieval, optimization and visualization. Regarding the nonfunctional requirements, which are generally harder to quantify, we can only emphasize that we have tested the application thoroughly and thus have handled a lot of edge cases that would otherwise result in errors. Furthermore, we tried our best at building a secure webapplication, for instance, by relying only on prepared SQL statements.

Our evaluation results show that our optimization methods can produce portfolios of acceptable performance. Overall, we believe our optimization tool to be helpful to both investors and analysts.

## List of Figures

4.1	UML component diagram showing the architecture of <i>EtfOptimizer</i> . . .	10
4.2	<i>EtfOptimizer</i> server deployment . . . . .	11
4.3	<i>EtfOptimizer</i> peer deployment . . . . .	12
4.4	Main loop extract of <code>justetf.com</code> crawler . . . . .	13
4.5	Loop requesting time series data via Eikon API (extract) . . . . .	14
4.6	Function for creating a dropdown list (extract) . . . . .	16
5.1	Dropdowns for the selection of the ETFs that should be considered . . .	24
5.2	Dropdowns and input fields for the selection of the optimization parameters	25
5.3	Visualization of the optimization results . . . . .	25
6.1	MSCI World against max sharpe optimized portfolio . . . . .	26



## List of Tables

5.1	Configuration Reference. Values marked with * must be set after first run of the CLI . . . . .	22
-----	--	----

# List of Abbreviations

**API** Application Programming Interface.

**CAPM** Capital Asset Pricing Model.

**CLI** Command-Line Interface.

**ETF** Exchange-Traded Fund.

**GUI** Graphical User Interface.

**ISIN** International Securities Identification Number.

**MPT** Modern Portfolio Theory.

**UI** User Interface.

# Bibliography

- [1] "C++ build tools installation tutorial." (Sep. 16, 2021), [Online]. Available: <https://drive.google.com/file/d/0B4GsMXCRaSSiOWpYQkstajlYZ0tPVkNQSElmTWh1dXFaykJr/view?resourcekey=0-HEezB2NFstz1GjKDkroJSQ>.
- [2] "Capital asset pricing model." (Oct. 9, 2021), [Online]. Available: <https://www.investopedia.com/terms/c/capm.asp>.
- [3] "Chrome." (Sep. 16, 2021), [Online]. Available: <https://www.google.com/chrome/>.
- [4] "Chrome driver." (Sep. 16, 2021), [Online]. Available: <https://chromedriver.chromium.org/downloads>.
- [5] G. Cornuejols and R. Tütüncü, *Optimization methods in finance*. Cambridge University Press, 2006, vol. 5.
- [6] "Cvxpy modelling language for convex optimization problems." (Oct. 9, 2021), [Online]. Available: <https://www.cvxpy.org/>.
- [7] "Dash." (Sep. 30, 2021), [Online]. Available: <https://dash.plotly.com>.
- [8] "Eikon api." (Sep. 30, 2021), [Online]. Available: <https://developers.refinitiv.com/en/api-catalog/eikon/eikon-data-api>.
- [9] J. Estrada, "Mean-semivariance optimization: A heuristic approach," *Journal of Applied Finance (Formerly Financial Practice and Education)*, vol. 18, no. 1, 2008.
- [10] "Etfoptimizer source code repository." (Sep. 28, 2021), [Online]. Available: <https://gitlab.lrz.de/etf-optimizing/etfoptimizer>.
- [11] "Expected returns." (Oct. 9, 2021), [Online]. Available: <https://pyportfolioopt.readthedocs.io/en/latest/ExpectedReturns.html#expected-returns>.
- [12] "Gurobi." (Sep. 16, 2021), [Online]. Available: [https://www.gurobi.com/documentation/9.1/quickstart\\_mac/software\\_installation\\_guid.html](https://www.gurobi.com/documentation/9.1/quickstart_mac/software_installation_guid.html).
- [13] "How is covariance used in portfolio theory?" (Oct. 15, 2021), [Online]. Available: <https://www.investopedia.com/ask/answers/041315/how-covariance-used-portfolio-theory.asp>.

- [14] O. Ledoit and M. Wolf, "Improved estimation of the covariance matrix of stock returns with an application to portfolio selection," *Journal of empirical finance*, vol. 10, no. 5, pp. 603–621, 2003.
- [15] O. Ledoit and M. Wolf, "Honey, i shrunk the sample covariance matrix," *The Journal of Portfolio Management*, vol. 30, no. 4, pp. 110–119, 2004.
- [16] H. Markowitz, "Portfolio selection," 1952.
- [17] R. A. Martin, "Pyportfolioopt: Portfolio optimization in python," *Journal of Open Source Software*, vol. 6, no. 61, p. 3066, 2021. doi: 10.21105/joss.03066.
- [18] "Optimization formulas." (Oct. 8, 2021), [Online]. Available: [https://palomar.home.ece.ust.hk/ELEC5470\\_lectures/slides\\_portfolio\\_optim.pdf](https://palomar.home.ece.ust.hk/ELEC5470_lectures/slides_portfolio_optim.pdf).
- [19] "Optimization formulas." (Oct. 9, 2021), [Online]. Available: <https://click.palletsprojects.com/en/8.0.x/>.
- [20] "Owasp top ten: Top 10 web application security risks." (Sep. 27, 2021), [Online]. Available: <https://owasp.org/www-project-top-ten/>.
- [21] "Plotly." (Sep. 30, 2021), [Online]. Available: <https://plotly.com/python/>.
- [22] "Postgresql database." (Sep. 16, 2021), [Online]. Available: <https://www.postgresql.org/download/windows/>.
- [23] "Python." (Sep. 16, 2021), [Online]. Available: <https://www.python.org/downloads/>.
- [24] "Record etf assets growth in 2020." (Sep. 30, 2021), [Online]. Available: <https://www.etf.com/sections/monthly-etf-flows/etf-monthly-fund-flows-december-2020?nopaging=1>.
- [25] "Refinitiv workspace." (Sep. 16, 2021), [Online]. Available: <https://www.refinitiv.com/en/products/refinitiv-workspace/download-workspace>.
- [26] "Scrapy documentation." (Sep. 29, 2021), [Online]. Available: <https://docs.scrapy.org/en/latest/>.
- [27] "Selenium with python." (Sep. 29, 2021), [Online]. Available: <https://selenium-python.readthedocs.io/>.
- [28] W. F. Sharpe, "Mutual fund performance," *The Journal of business*, vol. 39, no. 1, pp. 119–138, 1966.
- [29] W. F. Sharpe, "The sharpe ratio," *Journal of portfolio management*, vol. 21, no. 1, pp. 49–58, 1994.
- [30] "Sqlalchemy: Database urls." (Sep. 10, 2021), [Online]. Available: <https://docs.sqlalchemy.org/en/14/core/engines.html>.

## *Bibliography*

---

- [31] "Sqlalchemy: Dialects." (Sep. 10, 2021), [Online]. Available: <https://docs.sqlalchemy.org/en/14/dialects/>.
- [32] "Visual studio." (Sep. 16, 2021), [Online]. Available: <https://docs.microsoft.com/en-us/visualstudio/install/install-visual-studio?view=vs-2019>.
- [33] "XPath." (Sep. 29, 2021), [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/XPath>.