Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN TECNOLOXÍAS DA INFORMACIÓN

Application for the analysis of investment fund portfolios

Estudante: Miriam Breijo Fachal
Dirección: Paula María Castro Castro
José Pablo González Coma

A Coruña, xuño de 2020.

To my parents and my little sister Malena

Acknowledgements

First of all, I would like to thank my family for supporting me unconditionally and placing all their trust in me. I would especially like to name Emma, for how brave she is and the strength she has given me during these months.

Thanks also to my friends, thank you Sergio, for always being by my side. Of course, thanks to the great friends that I have met during the grade that have made these 4 years unbeatable.

Finally, many thanks to my project managers José Pablo González Coma and Paula María Castro Castro, without whom I could not have carried out this project. Thank you for all your attention and especially in this delicate situation that we have all experienced.

Abstract

An investment fund is a vehicle that collects the money of various citizens or companies to invest together.

With a single investment, you can have different stocks or bonds in your portfolio. In addition, when doing it together with other investors, the sum is greater and allows access to the best managers and at a lower cost than if we did it individually.

Once invested in a certain fund, we delegate to a manager the ability to decide where to invest. However, it is us, the investors, who will make the decision on which funds to invest in. Thanks to our application that gives us detailed and tailored information about the funds in our investment portfolio and our movements, making these decisions will be a much easier task.

Our application also displays charts that allow us to compare the funds in our portfolio at a glance. The calculation of the metrics and the returns (fiscal, financial dependent on our transactions and financial of the fund) is one of the main objectives of our project, since they will help us determine the future of our investments.

Resumo

Un fondo de inversión es un vehículo que reúne el dinero de diversos ciudadanos o compañías para invertirlo de modo conjunto.

Con una sola inversión, puedes tener distintas acciones o bonos en cartera. Además, al hacerlo junto con otros inversores, la suma es mayor y permite tener acceso a los mejores gestores y a un menor coste que si lo hiciésemos de modo individual.

Una vez invertido en un determinado fondo, delegamos en un gestor la capacidad de decidir dónde invertir. Pero somos nosotros mismos, los inversores, los que tomaremos la decisión de en qué fondos invertir. Gracias a nuestra aplicación que nos da una información detallada y a medida de los fondos de nuestra cartera de inversión y de nuestros movimientos, tomar estas decisiones será una tarea mucho más fácil.

Nuestra aplicación también muestra gráficas que nos permitirán comparar los fondos de nuestra cartera de un vistazo. El cálculo de las métricas y las rentabilidades (fiscal, financiera dependiente de nuestras transaccions y financiera del fondo) es uno de los principales objetivos de nuestro proyecto, ya que nos ayudarán a determinar el futuro de nuestras inversiones.

Keywords:

- Fund.
- Portfolio.
- Transaction.
- Currency.
- Net Asset Value.
- Profitability.
- Metrics.
- Graphics.

Palabras chave:

- Fondo.
- Cartera.
- Transacción.
- Divisa.
- Valor liquidativo.
- Rentabilidad.
- Métricas.
- Gráficas.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Making an investment can be key for a person or a company to achieve a significant economic gain in the short or long term. However, we are often reluctant to invest in financial products for various reasons: lack of adequate knowledge, lack of time to properly monitor the markets, not having sufficient savings to start investing... Meanwhile, others only invest to benefit from tax advantages or to increase their liquidity. The best solution for all these needs is an investment fund.

The basic principle of an investment fund is that it groups together a large number of investors, who receive a proportional title to the fund's assets in the form of units, according to the amount contributed. The fund manager uses the total assets contributed by the participants to buy and sell assets on behalf of the investors, forming a portfolio of securities. As this portfolio receives benefits, they are distributed among investors in the same proportion that investors have invested.

1.1 Goals

This project aims to obtain a desktop application in which users can obtain graphs, numerical data and results from one or more investment fund portfolios.

The purpose of this application is to simplify and facilitate how users manage the data published by websites or financial entities. In this way, through the generation of graphs and the calculation of indicators, the goal is to synthesize and visualize the data quickly and simply.

This desktop application contains the funds' files and user portfolios. For each fund, we track all its operations so that we can return real profitabilities, calculate the evolution over the desired time periods, compute useful ratios, etc. For each portfolio, it is possible to visualize the distribution of the invested capital, the total return on the portfolio, the more and less profitable funds, etc. And all this information displayed in an organized and detailed way.

Among the returns that we calculate, there is one that has a special importance, called fiscal return. With this calculation we trace the provenance of all the transactions of the selected fund, this is of great importance since in order to pay our taxes it is crucial to know if the deposit comes from cash or from money invested in another fund.

The measurements are customizable, allowing us to choose time periods. This type of functionality is not found in web services, since they only show pre-calculated data to users. In the same way, both the fiscal profitability and the profitability calculated according to our investments are two of our main objectives since they are functionalities not provided by any application or website.

The project does not include obtaining the necessary financial data to track. These data are provided by the corresponding financial entity. However, the application allows import funds with their net asset values from an url where we find a wide range of funds, as well as the possibility of importing currencies from a csv file.

1.2 Motivation

Unfortunately, with the situation we are experiencing socially and economically, there is a great instability in the stock market. This instability caused by COVID-19 opens the door to new scenarios in the global economy [1] [2]. There is much fear on the part of investors about how this situation will progress, despite being impossible to predict, we hope that our application can help make the best decisions with regard to our purchases and sales of shares.

In addition to the reason explained above, we should mention that to this day there is no application that meets all the features offered by our proposal, so we foresee a great future for this application.

Because it responds to this growing demand that we have mentioned and because I have always been interested in the stock market, I have chosen this project offered by my tutors.

1.3 Work Structure

Below we will briefly describe the chapters in which the report has been structured.

- **Chapter 1. Introduction**: We describe our objectives, motivations and memory structure.
- **Chapter 2. Theoretical Bases**: We explain what investment funds consist of, how they are classified and what criteria we should use when investing. We will also define ratios that will help us make these decisions and what operations will occur between our funds and portfolios.

- **Chapter 3. State of the Art**: We evaluate the tools that currently exist in the market that are related to our project.
- **Chapter 4. Technological Bases**: We explain what tools we have used to develop our application.
- **Chapter 5. Methodology**: We describe the chosen methodology, the elements that compose it and how we have applied it to our project.
- **Chapter 6. Planning and Cost Estimate**: Project planning, follow-up and the estimated cost of our application are defined.
- **Chapter 7. Requirements Analysis**: We will describe the functional and non-functional requirements of our app.
- **Chapter 8. Design**: We explain the chosen architecture and how we have designed the database.
- **Chapter 9. Development**: We describe the implementation of the database and the service. We will also show the interface and its use, along with an explanation of its development.
- **Chapter 10. Tests**: We explain how we have verified the correct operation of our application and that the requirements are met.
- **Chapter 11. Conclusions and Future Work**: We will describe the conclusions obtained and how we could improve or continue implementing it so that our project improves and grows.

Chapter 2

Theoretical Bases

In order to carry out this project, the first step was devoted to the search for information about the world of finance, more specifically about investment funds, in order to know how they work, how to calculate their returns and ratios and what criteria should we use to invest.

2.1 Investment Funds

To begin we will define what a fund is, how it works and the elements that intervene on it.

An **investment fund** is a capital made up of the sum of monetary contributions done by several people [3]. This capital will be invested in a series of assets with the objective of obtaining the maximum possible profitability. Depending on the evolution of these assets, the fund will throw positive or negative results, which will be distributed among each investor according to the proportion that their investment represents over the total assets of the fund. Each fund is identified by an International Securities Identification Number (ISIN) [4]. This code uniquely identifies a movable value on an international level.

Investment funds are divided into proportional parts called **holdings** and their owners are called **participants**. The number of participants is not fixed, but depends on their purchases and sales. Its value, called the **Net Asset Value (NAV)** [5], is daily calculated as follows:

$$\text{NAV} = \frac{\text{Fund Assets} - \text{Fund Liabilities}}{\text{Number of Units Outstanding}}, \tag{2.1}$$

where

- *Fund Assets* includes the total market value of the fund's investments, cash and cash equivalents, receivables and accrued income;
- *Fund Liabilities* are money owed to the lending banks, pending payments and a variety of charges and fees owed to various associated entities, and

- *Number of Units Outstanding* is the total number of shares outstanding.

This value depends, therefore, on the daily evolution of the values that make up the fund assets and will be one of the fundamental indications that the application will use when making the records of the different funds.

In a fund, investment decisions are made by a fund manager that manages and represents the fund, while the function of custody and monitoring assets is carried out by the depositary, usually a financial institution. Normally the manager charges a series of management fees that are subtracted from the fund, which decreases the NAV of each holding.

In the next section we will explain what type of operations we can perform on them.

2.2 Operations and Fund Tracking

In this section we will talk about the operations of subscription, reimbursement and transfer of an investment fund. We will also explain how to track your profitability.

2.2.1 Operations

The method to make an investment in a fund is the **subscription** [6] of holdings. The managing entity issues a series of them and each investor obtains as many as the result of dividing the capital invested by the NAV applicable to the operation. Normally, the applicable NAV is the same as the day of the request or the next day. Some funds may be subject to subscription fees of up to 5% of the investment.

If an investor wants to recover his money, he must request a **refund** [6] of all or part of his/her holdings, receiving the result of multiplying the NAV of the participation by the number of holdings he/she wants to reimburse. The applicable NAV is the same as in the previous case. The period in which the investor receives his money is a maximum of 3 to 5 days, and said reimbursement may have a commission of up to 5%. The investor will know the result of the investment (positive or negative) when the refund is paid.

In the case of wanting to transfer from one fund to another, there is a refund of the first and the immediate subscription of the second. The units that we sell in each refund are the first ones that were bought and, of course, have not been sold yet.

There are four parts involved in a transfer:

- **Source fund**: Fund in which the investment is maintained before the transfer.
- **Target fund**: Fund in which you want to invest the capital that is repaid from the source fund.
- **Source entity**: The one that markets or manages the source fund.

- **Target entity**: The one that markets or manages the target fund.

Each fund will use its own currency and in the event that the source fund and target fund of a transaction have different ones, we must apply the currency exchange [7].

However, since it is a reimbursement and subscription operation, the respective commissions that have both funds established must be paid.

In addition to pay commissions, tax capital gains [8] are something for which we must pay taxes. However, in the case of transactions between funds, these capital gains will only be taxed when we withdraw the money invested from the fund, not when we make operations between them.

The result is not perceived effectively until the refund of the holdings takes place and it will be at that moment in which the participants must pay taxes for the result of their investment.

2.2.2 Fund Tracking

The process of monitoring an investment fund can be carried out mainly through two sources:

- The documentation provided by the managing entity, since it is mandatory that the participants be provided with periodic information about the evolution of their investments.
- The disclosure of data on investment funds provided by newspapers or several Internet portals. From this last source we will obtain the necessary data for the initial operation of the application.

The following points will focus on knowing the different types of funds on which we will carry out our operations and the criteria that should be used for our choice.

2.3 Fund Types

We are going to explain several types of investment funds [9], describing their characteristics and the investor profile for which they are the most suitable. This knowledge will help us making decisions about them.

- **Fixed-Income Funds**: These funds are those where the majority of the capital is invested in fixed-income assets, with changes to interest rates being the factor that will influence on the evolution of these funds the most. The shorter the maturity period for the assets the fund is investing in, the lower the risk and therefore the lower the return.

  This kind is especially suitable for investors with a conservative profile, who are willing to accept lower returns in exchange for greater peace of mind.

- **Variable-Income Funds**: Also known as equity funds, the majority of the capital is invested in variable-income assets (stocks). In contrast to the situation with fixed-income funds, variable-income funds offer higher potential returns because the risk assumed is also higher.

  Subcategories are normally established within equity funds, depending on the market being invested in (Spain, Eurozone, USA, etc.), depending on the sectors being invested in (technology, financial, etc.), or depending on other characteristics of the securities being invested in (size of the company, etc.).

  Because of their characteristics, variable-income funds are recommended for a more resolute investor profile. Greater risks are assumed during investment, but this can bring with it higher potential returns.

- **Mixed-Income Funds**: These funds diversify the investment by investing part of their capital in fixed-income assets and the rest in variable-income assets. It is especially important to know these proportions, since they will determine how much risk is associated with the fund and therefore the size of the potential returns.

  Mixed-income funds are products designed for all types of investor profiles, from the most conservative to the most resolute, depending on the percentages dedicated to fixed-income and variable-income.

- **Guaranteed Funds**: These funds guarantee, up to a specific date, preservation of the capital initially invested. However, not all of these funds guarantee that the investor will receive additional returns. In general, they tend to require investors to keep their money invested for a long period of time.

  The risk associated with funds of this type is quite low, which means that they are suitable for investors with a conservative profile.

So far we have discussed the risks we will take depending on what type of fund we invest in, but in the next two types we will discuss what will be done with the benefits.

- **Distribution Funds**: These funds periodically distribute dividends to their investors (monthly, quarterly, twice-yearly, or yearly). The amount of these payments will depend upon the dividends distributed by the companies in which the fund has invested. This provides liquidity for the investor, but taxes must also be paid on the dividends received.

- **Accumulation Funds**: These funds do not distribute dividends to their investors. Instead, the manager reinvests the dividends that the companies pay out back into the same fund. This means that the fund's net asset value grows progressively.

2.4 Investment Selection Criteria

As we have seen in the previous section, there are several types of investment funds adapted to different needs. When choosing a particular fund, there are some ratios and indicators that can help determine which one is best suited to investor preferences.

Normally, when selecting a fund, the investor should consider what is his capacity to assume losses (because the greater the risk, the greater the profitability). We will also observe the time horizon during which you wish to keep the investment, because depending on the fund's policy, it may be advisable to be willing to keep the investment for a certain period of time.

In addition, we should take into account the fees charged to investment funds, since they affect profitability. It is possible for a fund to apply different types of commissions to the different types of shares it issues.

We must also consider the historical behaviour that a fund has had over time. It is important to know the returns obtained in the past, although this does not mean that a similar line is followed in the future. In the application to be developed, historical records of the returns referring to a certain period (quarter, semester ...) will be included; so that when comparing different funds, the returns in the same periods can be checked. It is worth mentioning that it is necessary that the funds follow the same investment policy in order for the comparison to be significant.

It is possible that during the life of a fund it will change its investment policy and even the management group, so when consulting past returns, it should be taken into account that they may have changed. It is important to know the date of said change and take into account only the returns from that moment.

2.5 Profitabilities

Also, the fund's profitability is a very important metric. This is calculated using the percentage difference between the NAV on the date of purchase of the holding (subscription) and the date of sale (refund), as follows,

$$\text{Profitability} = \frac{\text{Final NAV} - \text{initial NAV}}{\text{Initial NAV}} \times 100. \tag{2.2}$$

However, it should be taken into account that the previous percentage does not reflect annual profitability, but the one obtained in the period between the date of the initial value and that of the final value. To obtain the annual rate of return it is necessary to annualize the profitability [10] using the following formula,

$$\text{Annual profitability} = \left(\left(\frac{\text{Final NAV}}{\text{Initial NAV}} \right)^{365/d} - 1 \right) \times 100, \tag{2.3}$$

where d is the number of days between the initial and final NAVs.

We will not only calculate the fund's return, which is independent of our movements. We will calculate two profitabilities that will depend on the investment decisions (refunds and deposits) that we have performed.

- **Financial Own Profitability**: This is the profitability calculated not only from the initial and final NAV of the selected range, but also taking into account the purchases and sales made in the selected fund.

$$\text{Financial Own Profitability} = \frac{\text{NAV} \times \text{holdings} + \text{refunds} - \text{deposits}}{\text{deposits}}, \tag{2.4}$$

  where

  - *NAV* is the value of the last NAV that we found;
  - *holdings* are the units that we currently have in the fund;
  - *refunds* are the shares that we have sold, and
  - *deposits* are the number of units we have purchased.

- **Tax Return**: To calculate this return, we will need to track the origin of the money in each transaction. This supposes a recursive search, where the money can come from a cash purchase, from a past transaction or from several (in the case of coming from transactions we will have to continue tracking each one). Once the provenance of each of the transactions that make up the fund is found, we will be able to see the return value associated with the number of units acquired in the fund.

  As we explained at the beginning of the chapter, having the tax return calculated for each fund will be of great help when it comes to investing.

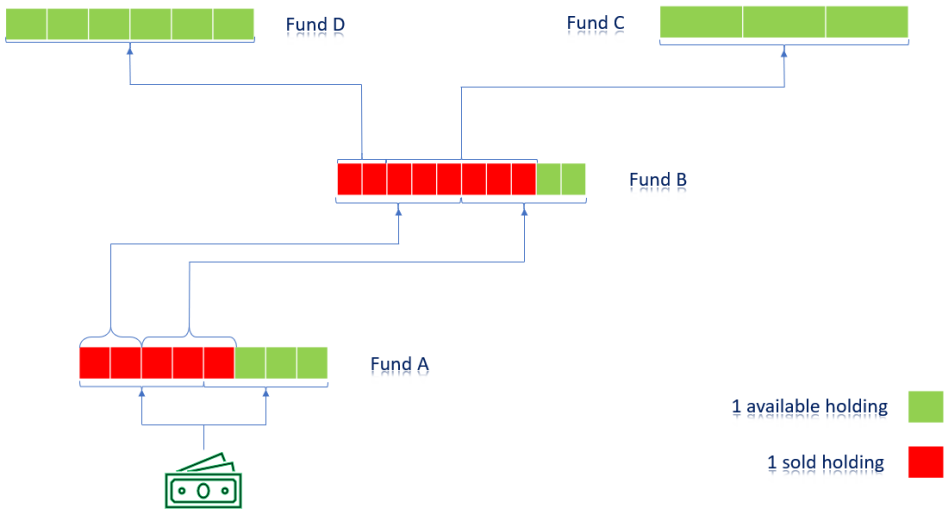


Figure 2.1: Diagram representing the flow of transactions to be traced to calculate tax profitability.

2.6 Ratios or Metrics

Now, we will see a series of ratios or metrics that will help us finding an investment fund that suits our needs, depending on our risk profile.

- **Sharpe Ratio** [11] This ratio helps investors understand the return of an investment compared to its risk. It is calculated as follows,

$$\text{Sharpe Ratio} = \frac{R - R_f}{\sigma}, \tag{2.5}$$

  where

  - R profitability of our fund;
  - R_f is the risk-free rate, and
  - σ is the standard deviation of the **fund's excess** return.

- **Beta** [12] A beta coefficient is a measure of the volatility, or systematic risk, of an individual stock in comparison to the unsystematic risk of the entire market. It is calculated as follows,

$$\beta = p_{im}\frac{\sigma_i}{\sigma_m}, \tag{2.6}$$

where

- p_{im} is the correlation between the fund and the benchmark;
- σ_i is the standard deviation of the fund, and
- σ_m is the standard deviation of the benchmark (IBEX 35 for example).

- **Treynor Ratio** [13] Also known as the reward-to-volatility ratio, is a performance metric for determining how much excess return was generated for each unit of risk taken on by a portfolio.

$$\text{Treynor Ratio} = \frac{R - R_{\text{f}}}{\beta}, \tag{2.7}$$

where

- R fund return;
- R_{f} is the risk-free rate, and
- β is the beta of the fund.

- **Jensen's alpha** [14]: It is used to determine the abnormal return of a security or fund of securities over the theoretical expected return. It is given by

$$\alpha_{\text{J}} = R - [R_{\text{f}} + \beta \cdot (R_{\text{M}} - R_{\text{f}})], \tag{2.8}$$

where

- R is the realized return;
- R_{M} is the market return;
- R_{f} is the risk-free rate, and
- β is the beta of the fund.

- **Correlation coefficient** [15]: It measures the meaning and similarity in the relationship between the fund and the market. It oscillates between 1 and -1. Positive indicates the fact that the market and the fund fluctuate in the same direction. This is an important ratio in the construction of investment fund portfolios, since the positive effect of diversification depends on the correlation between the different assets of a portfolio. It is defined as follows,

$$\rho_{X,Y} = \frac{\text{cov}(X, Y)}{\sigma_X \sigma_Y}, \tag{2.9}$$

where

- $\text{cov}(X, Y)$ is the covariance of fund and market;
- σ_X is the standard deviation of the fund, and
- σ_Y is the standard deviation of the market.

- **Maximum Drawdown (MDD)** [16]: It is the maximum observed loss from a peak to a trough of a fund, before a new peak is attained. Maximum drawdown is an indicator of downside risk over a specified time period. It is defined as follows,

$$\text{MDD} = \frac{\text{Trough Value} - \text{Peak Value}}{\text{Peak Value}}. \tag{2.10}$$

Chapter 3

State of the Art

In this chapter we review the software similar to our application. We discuss the characteristics of these tools in order to appreciate the novelties that our project brings.

3.1 MorningStar

Morningstar, Inc. [17] is a global financial services firm headquartered in Chicago founded by Joe Mansueto in 1984. It provides an array of investment research and investment management services.

Morningstar's research and recommendations are considered by financial journalists as influential in the asset management industry, and a positive or negative recommendation from its analysts can drive money into or away from any given fund.

If we use the search engine and enter the ISIN, this famous website, MorningStar, will return various data on the fund: category, Net Asset Value, benchmark... If we want to add a fund to the portfolio as we would in our app, we must be members, otherwise there is no possibility.

It also provides us with an assessment that they themselves make of the fund, some graphs, how it is managed and a link to useful definitions that will help us if we do not know the terminology.

If we access the *Profitability* tab, it tells us about annual and accumulated returns, but it does not inform us about the own returns for eligible periods, and do not provide the tax return.

On the Home page we find news, rankigs, etc. that will help and encourage us to make new investments. In our example, it talks about the ranking of Spanish managers in May, the rebound that is taking place in the stock market and whether it is appropriate to buy now and the top 10 companies with cheapest values but that are increasing their competitiveness.

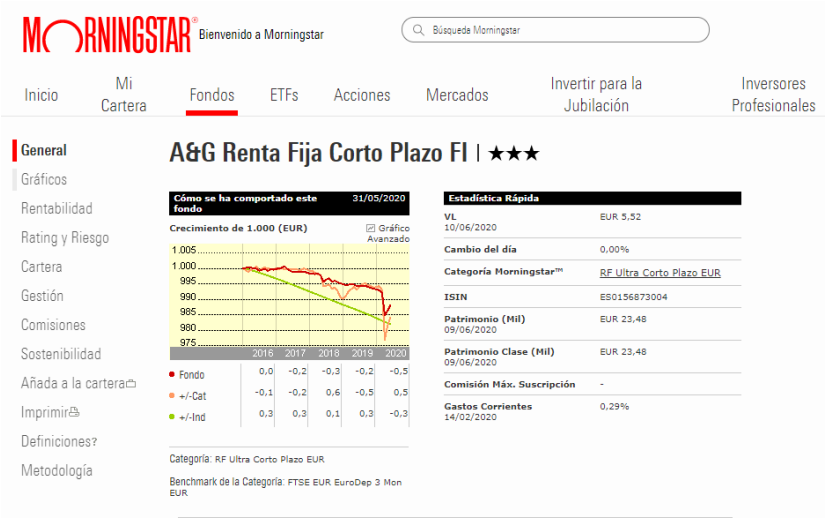


Figure 3.1: Result of searching for an ISIN at morningstar.com.

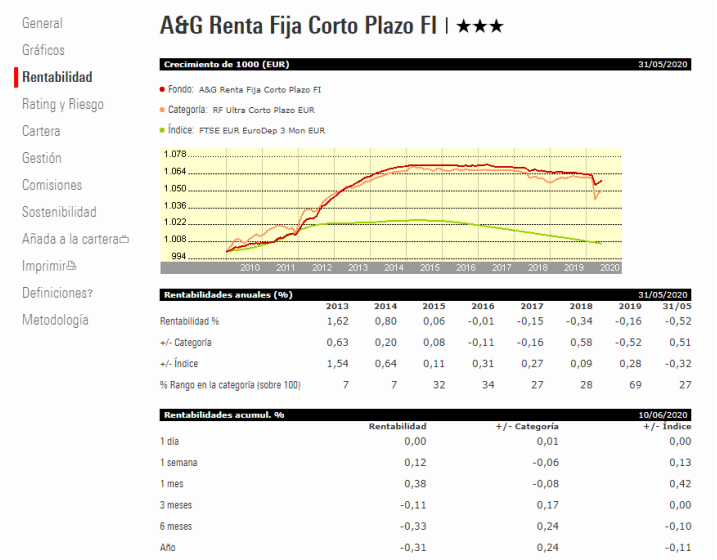


Figure 3.2: Returns provided by MorningStar about the chosen fund.

3.2 Banks

Of course the banks also offer us funds to invest in, although there are some funds that they have reserved for premium or VIP clients (they will invest much larger amounts).

For example, there are 132 Spanish fund managers where the average investment of the participants exceeds one million euros, according to calculations based on data from Inverco, the industry association [18]. And if we make the cut in the 100 000 euros of average participation, which is already a considerable amount of money, we find another 500 funds.

Any of the large banks in Spain (BBVA, CaixaBank, Santander, Bankinter, Sabadell, Bankia...) offers us a variety of funds. We are going to show the example of the Santander bank. It has a search engine where the funds that it markets appear, where we can view some of the most important information: returns calculated for the fund (not for its specific operations), continuing without showing the tax return and a history of NAVs.

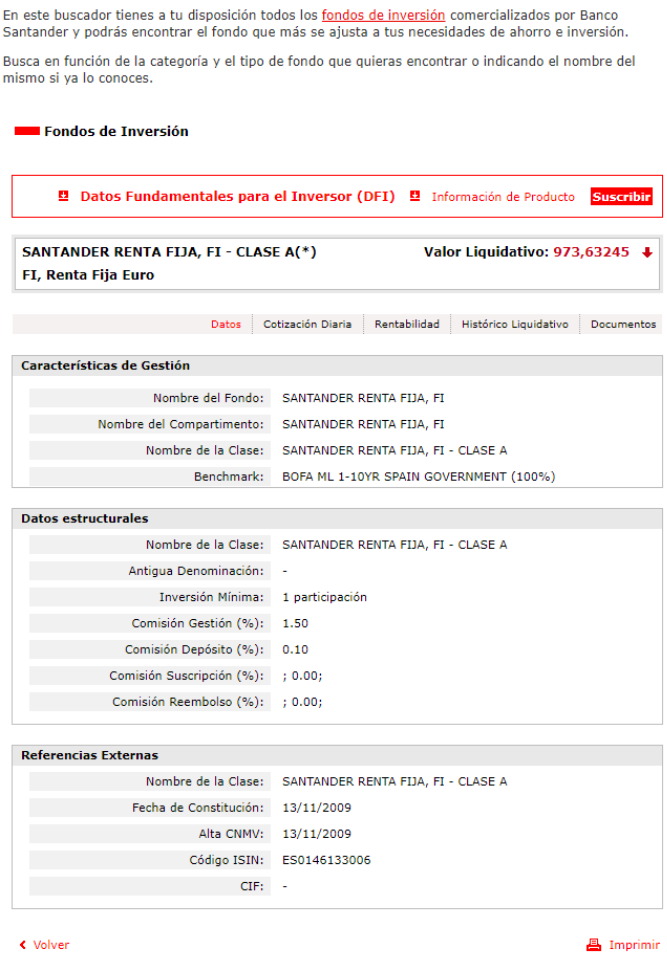


Figure 3.3: Result returned by the search for a fund in Santander bank.

3.3 Applications To Manage Our Investments

There are some applications that allow us to be aware of our investments, some only offer us information on the financial market, where it has nothing to do with our specific investments. For example, Fonditus [19] is an app where we can add the funds that interest us and where we can see the history of their NAVS, calculation of performance, etc.

On the other hand, there are some apps that allow us to invest from themselves or to simulate our investments, either by copying investments from professionals or by offering daily offers chosen by its own algorithm. This is the case of Finizens [20], which are specialists in passive investment (trying to replicate a certain index instead of beating it, as active management attempts), it shows information about our funds but not the detailed information of our transactions as it can be the dates of refund and deposit.

3.4 Conclusions

In this section we will show a comparative table of the functionalities that MorningStar incorporates, the Santander bank and the two applications that we have explained with the application that we have developed (**FundApp**).

	MorningStar	**Santander**	**Fonditus**	**Finizens**	**FundApp**
Fund information	✓	✓	✓	✓	✓
NAVs history plot	✓	✗	✗	✓	✓
Charts comparing portfolio funds	✗	✗	✗	✗	✓
Allow to invest or simulate investments	✗	✓	✗	✓	✓
Detailed transactions	✗	✗	✗	✗	✓
Returns of our fund	✓	✓	✓	✓	✓
Profitability tailored to our movements	✗	✗	✗	✗	✓
Fiscal return	✗	✗	✗	✗	✓

After searching for information on numerous occasions, the conclusion is that the applications that manage our portfolio are scarce and we have not found any that provide us specific information about our investments. Most of the opinions that we have found only recommend seeing the MorningStar page, they do not speak of the real usefulness of any application.

Chapter 4
Technological Bases

This chapter describes the tools and technologies that have been taken into account for the development of the project.

4.1 Version Control Tools

The main alternatives for the project repository are the following:

- **GitHub**: GitHub is a Git repository hosting service, but it adds many of its own features. While Git is a command line tool, GitHub provides a Web-based graphical interface. It also provides access control and several collaboration features, such as wikis and basic task management tools for every project.
- **GitLab**: It is a repository manager which lets teams collaborate on code. Written in Ruby and Go, GitLab offers some similar features for issue tracking and project management as GitHub.

GitHub has been chosen as the version management system of the project over its main alternative, GitLab, since it is the most used platform and it has been easier for us to make project collaborators.

4.2 Project Management Tool

- **Microsoft Project**: It is a project management software product. Microsoft Project is designed to assist a project manager in developing a schedule, assigning resources to tasks, tracking progress, managing the budget, and analyzing workloads.

Because it is one of the most widespread software for this task and because we are already familiar with it, it has been our choice.

4.3 Software Modelling Tools

The main alternatives for software modeling are the following:

- **Visual Paradigm Community Edition**: It is a tool for application development using Unified Modeling Language (UML) recommended for the application and monitoring of the Rational Unified Process (RUP). It provides assistance to perform the analysis, design, use cases and UML models of the project.
- **Dia**: It is an application for creating diagrams. It is conceived in a modular way, with different packages of shapes for different needs.

Visual Paradigm Community Edition has been chosen for the realization of most project diagrams, such as UML or those of the relational model because it provides higher quality diagrams.

4.4 Project Development Tools

The main alternatives for Integrated Development Environment (IDE) are the following:

- **Eclipse IDE 2019-12**: It is an open source software development platform based on Java. It is valid for almost any language, although Java is the most used. It also provides a series of plugins for version control and frameworks for the development of graphic applications.
- **NetBeans IDE**: It is a free integrated development environment that allows applications to be developed from a set of software components called modules. NetBeans is designed primarily for the development of Java applications and contains a framework that simplifies the development of Java Swing desktop applications.

Eclipse has been chosen as the development platform for this project instead of NetBeans because it has been the platform used in most subjects of the computer engineering degree, so its operation is more familiar.

The main options such as project management software are the following:

- **Apache Maven**: It is a project management software developed by Apache. It is based on the concept of Project Object Model (POM). Maven allows to manage dependencies, modules, components and the order of construction.
- **Gradle**: It is a software construction tool that combines Ant's flexibility with Maven's conventions that uses a Directed Acyclic Graph (DAG) to determine the order in which tasks should be executed.

Maven has been chosen for the same reason as the previous software, it is the most used and known manager during our undergraduate studies.

4.5 Database Tools

The main alternatives regarding database management software are the following:

- **MySQL** [21]: It is a Relational Database Management System (RDBMS) developed by Oracle Corporation and is considered the most popular open source database in the world. MySQL provides a very fast database in reading but it might incur into problems in environments of high concurrence in the updates.
- **PostgreSQL**: It is an object-oriented relational database management system, which provides high concurrency and a wide variety of natives.

It has been decided to use MySQL because our application will mainly use simple queries, usually reading, and MySQL is oriented to this type of tasks providing better performance than its competitor. MySQL itself will generate the Entity-Relationship diagrams.

The main options as an object-relational mapping framework are:

- **Hibernate** [22]: It is an object-relational mapping tool for the Java programming language. Its use facilitates the mapping of attributes between a traditional relational database and the object model of an application, using files called eXtensible Markup Language (XML) or annotations in the beans of the entities to establish these relationships.
- **Apache iBATIS**: It is a persistence framework which automates the mapping between SQL databases and objects in Java, .NET... In Java, the objects are POJOs (Plain Old Java Objects). The mappings are decoupled from the application logic by packaging the SQL statements in XML configuration files. Other persistence frameworks such as Hibernate (named before) allow the creation of an object model (in Java, say) by the user, and create and maintain the relational database automatically. iBATIS takes the reverse approach: the developer starts with a SQL database and iBATIS automates the creation of the Java objects.

Finally we decided to use Hibernate because it has a great integration with our IDE and we already have prior knowledge.

4.6 Graphical Interface Development Tools

The main alternatives such as graphic libraries for Java are the following:

- **Swing**: It is a Graphical User Interface (GUI) widget toolkit for Java. It includes widgets such as text boxes, buttons, drop-down boxes, tables...
- **JavaFX** [23]: It is a set of graphics and media packages that enables developers to design, create, test, debug, and deploy rich client applications that operate consistently across diverse platforms. JavaFX is intended to replace Swing as the standard GUI library for Java SE, but both will be included for the foreseeable future.

It has been decided to use JavaFX because it has richer GUI components and fast UI development with screen builder [24].

4.7 Graphics Creation Tool

The main option as a library for creating graphics is the following:

- **JFreeChart**: It is an open-source framework for the programming language Java, which allows the creation of a wide variety of both interactive and non-interactive charts. JFreeChart supports a number of various charts, including combined charts: X-Y charts (line, spline and scatter), pie charts, Gantt charts, bar charts (horizontal and vertical, stacked and independent)... It is possible to place various markers and annotations on each plot.

This library is by far the most used by Java applications that use graphics, so it gives us more possibilities than we could find in most web interfaces.

4.8 HTML Parser Tool

The main alternative as an Application Programming Interface (API) for parse HTML to Java is:

- **Jsoup** [25]: It is a Java library for working with real-world HTML. It provides a very convenient API for fetching URLs and extracting and manipulating data, using the best of HTML5 DOM methods and CSS selectors.

4.9 Testing Tool

The main option as a test framework is the following:

- **JUnit**: It is a unit testing framework for the Java programming language, allowing to perform tests in a controlled way and evaluate the operation of each of the methods of the classes. JUnit is linked as a JAR at compile-time.

4.10 Text Edition Tool

- **LaTeX**: It is a high-quality typesetting system; it includes features designed for the production of technical and scientific documentation. LaTeX is the de facto standard for the communication and publication of scientific documents.

This is the chosen option because it is the editor that we must use to write the project, being also a very handy tool and already known to us.

Chapter 5

Methodology

"A software development methodology is kind of like a cooking recipe. Like a recipe tells you how to cook a meal, a software development methodology tells you how to build a software product." So one of the most important decisions of the project is which methodology we are going to use.

Waterfall methodology is the traditional one and is a Liner Sequential Life Cycle Model whereas Agile is a continuous **iteration** of development and testing in the software development process. Also, Agile is completely based on the **incremental** progress. Therefore, both the client and the team exactly know what is complete and what is not. This reduces risk in the development process.

We have chosen the iterative and incremental approach since it provides us more adaptability and assures that quality of the development is maintained.

When we say a methodology is Agile, it means that it follows the set of values and principles recorded in a namesake manifesto. However, there is more than an unique way to implement it. In fact, there are many different types of Agile methodologies that you can choose from when organizing your project. There are Kanban, Extrem Programming (XP), Feature-Driven Development (FDD)... But we are going to choose the most widely spread that is Scrum, one of the frameworks that revolutionized the software development industry in recent years.

5.1 Scrum

Scrum [26] is a framework that helps teams work together. Much like a rugby team (where it gets its name) training for the big game, Scrum encourages teams to learn through experiences, self-organize while working on a problem, and reflect on their wins and losses to continuously improve.

5.1.1 Artifacts

First, we should define what artifacts mean. They are something that we make, like a tool to solve a problem and there are three types:

- **Product Backlog**: It is an inventory that contains any type of work that needs to be done on the product: requirements, use cases, tasks and dependencies. It is the main source of information about the product in Scrum, a list, in any format, that contains all the requirements that we need to implement in the product. It must be exclusively managed by the Product Owner.
- **Sprint Backlog**: This is a list of items to work on during the current Sprint cycle and is managed by the Development Team.
- **Increment** or **Sprint Goal**: It is the sum of all the tasks, use cases, user stories and any element that has been developed during the Sprint; that will be made available to the end user in the form of software, providing business value to the product being developed. Briefly, it is the result of the Sprint.

5.1.2 Sprint

Sprint is the actual time period when the scrum team works together to finish an increment. The duration of a Sprint is determined by the minimum period in which a Development Team can generate value. Besides, Sprint is the container for all other Scrum events.

Ceremonies or roles

- **Sprint Planning**: It is a meeting that takes place at the beginning of each Sprint where the entire Scrum team participates. It is used to inspect the Product Backlog and for the development team to select the Product Backlog Items to work on during the next Sprint. These Product Backlog Items are what will make up the Sprint Backlog.
- **Daily Scrum**: The goal of this daily super-short meeting is for everyone on the team to be on the same page, aligned with the Sprint goal, and to get a plan out for the next 24 hours.
- **Sprint Review**: It is the meeting that occurs at the end of the Sprint. Here the Product Owner and the Development Team present the finished increment to stakeholders for their corresponding inspection and adaptation.
- **Sprint Retrospective**: The idea is to create a place where the team can focus on what went well and what needs to be improved for the next time, and less about what went wrong.

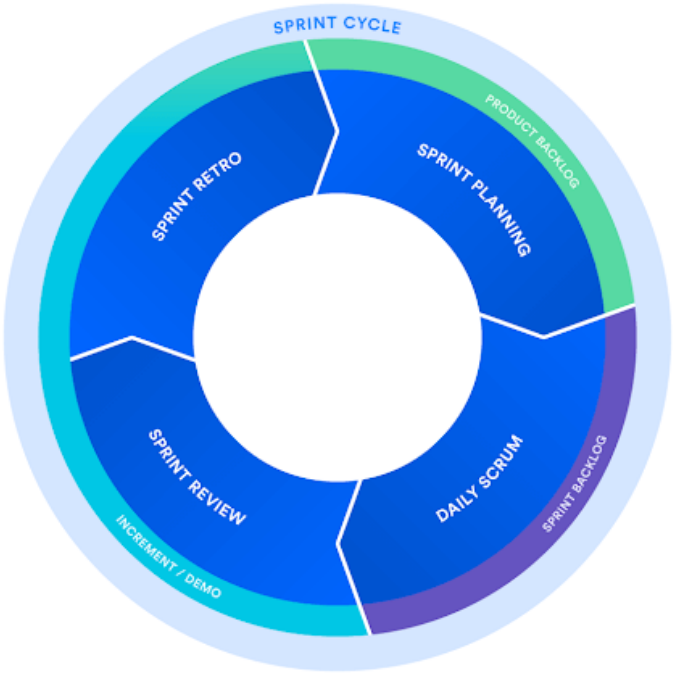


Figure 5.1: Sprint cycle.

5.1.3 Roles

Each of these three roles [27] has different responsibilities and must be held accountable differently, both among themselves and for the rest of the organization. The sum of all the roles is what we call the Scrum Team.

- **Product Owner**: He or she is in charge of optimizing and maximizing the value of the product, being the person in charge of managing the flow of product value through the Product Backlog. With each Sprint, the Product Owner must make a development investment that must produce value. Marking the Sprint Goal clearly and agreed with the development team makes the product constantly increases its value.

- **Scrum Master**: He/she has two main functions within the framework: managing the Scrum process and helping to remove impediments that may affect product delivery. The Scrum Master must be responsible for ensuring that this framework is carried out, transmitting its benefits to the organization.

- **Development Team**: They are in charge of developing the product, self-organizing and self-managing to deliver a software increment at the end of the development cycle. The Sprint Goal is made from the selected Product Backlog elements (Sprint Backlog) during Sprint Planning.

5.2 Scrum applied to our Project

In this section it is shown how we have implemented Scrum in our project.

5.2.1 Product Backlog

This part will list the functionalities of the different intermediate products that make up this project.

0. Research:
    - Search for information about investment funds with their corresponding metrics, criteria and transactions.
    - Review and search for technologies and tools for project development.
1. Requirements analysis:
    - Define the functional and non-functional requirements of our application.
    - Illustrate the requirements as use cases and what characters or entities will interact with them, actors.
2. Data modelling design:
    - Decide what entities we need, what attributes and what relationships.
    - Avoid redundancy and determine what attributes to put or not depending on whether it would be better to permanently have it in the DB or dynamically calculate it each time.
3. Database creation and start of layered development:
    - Create tables with their corresponding attributes (columns).
    - Create model layer with their DTOs.
    - Define service interface.
4. Deepening in the development and design of the graphical interface:

- Implementation of the service comprising all the logic. Therefore, here we develop the calculations for all the necessary ratios and metrics.
- Controller creation.
- Design and constitution of graphical interface.
- Graphics generation for portfolios and funds.

5. Work disclosure:

   - Reflect conclusions of our project and the result of our application.

5.2.2 Sprints

The project is divided into Sprints as indicated by the methodology we have chosen and they will be adapted to the duration of our project and the difficulty that prevails in each phase.

Sprint 0: Research

In this initial phase we acquire the necessary theoretical knowledge about investment funds and find out which technologies are most suitable for our project.

Once this Sprint is finished, we already know the costs of the resources and, with the concepts we have learned, we can undertake the requirements analysis phase.

Sprint 1: Requirements analysis

Defining the specific functionalities that our system must fulfil is the first step before being able to develop. We have to identify functional and non-functional requirements. Once they are identified, we will illustrate them in their corresponding use cases with a sufficient level of detail.

Sprint 2: Data modelling and architecture design

Before beginning development, we need to identify the entities and the relationships among them. We have to be very careful not to generate redundancies or inconsistencies. Also deciding whether an attribute will be persistent or dynamically calculated can be a crucial decision in our project.

A design pattern is a reusable way to solve a common problem. It describes the problem, the solution, when to apply the solution, and its consequences. So we have to reason and analyse which patterns will our architecture follow.

Sprint 3: Database creation and start of layered development

With the design obtained in the previous Sprint, we can now get down to work with the creation of the database. Once finished, we can start with the model and its corresponding DTOs. Finally in this Sprint, we will generate the service interface with the corresponding methods that will implement the logic.

Sprint 4: Deepening in the development and design of the graphical interface

As we have already said, it is time to implement logic. In the previous point we have defined the methods and now we have to do the relevant calculations and operations.

Finally, we will design and build the graphical interface. The creation of graphics that are as representative as possible of our funds and portfolios is highly relevant in this application in particular.

Sprint 5: Work disclosure

Although the writing of the report has been progressive (from research, analysis, design to development), in this last Sprint, we will explain the conclusions obtained from our project and what is the result of our application.

Tests

It should be noted that there will be a test phase in all Sprints that include design or development. This will prevent us from encountering a cluster of errors at the end point for which we do not even know their source.

Meetings

Throughout the project, we have been holding meetings between the three project members. At first, more dispersed and as the project began to be designed and developed, the meetings began to be held weekly.

Chapter 6

Planning and Cost Estimate

In this chapter we will include the planning of our project and its estimation in cost. We will also present a follow-up of such a planning.

6.1 Initial Planning

First, we can see the tasks with their corresponding Gantt diagram (see Fig. 6.1) where we show the six Sprints constituting our project.

In the previous chapter (see Chapter 5) we have explained our Sprints in detail, so we will also see their planning with the specific tasks of the first three Sprints (see Fig. 6.2) and of the remaining ones (see Fig. 6.3). We have adapted the duration of each Sprint to the number of tasks that make it up and its difficulty. For example, the Sprint that includes part of the service implementation and the interface has a much longer duration programmed than the Sprint of the database design or that of the conclusions.

Also, in Sprints 3 and 4, which include the total of the development, a large number of tests will be produced and therefore requires a longer duration (see Chapter 10).

6.2 Resources and Cost Estimate

Project resources are people, capital and/or material goods required for the successful execution and completion of a project. They should be assessed and allocated before a project begins. Poor resource planning can result in running out of resources midway through a project, delaying deadlines, and delaying delivery of the final product or service.

6.2.1 Human Resources

As we have explained, there are three roles defined in Scrum that are distributed among the three members of the project as follows:

- José Pablo González Coma as Product Owner.
- Paula María Castro Castro Coma as Scrum Master.
- Miriam Breijo Fachal as the unique member of the Development Team. Although it usually consists of from 3 to 9 professionals, this is a methodology adaptation to the context of this work.

Of course, depending on the role you have and your responsibilities, the quantity you will earn per hour is different. Here, we have a table with the estimated costs for each one:

	Product Owner	Scrum Master	Development Team
José Pablo González Coma	30€	-	-
Paula María Castro Castro	-	30€	-
Miriam Breijo Fachal	-	-	25€

Table 6.1: Hourly cost of human resources.

Considering that the Development Team works an average of 5 hours a day and both Scrum Master and Project Owner work 1 hours a day, in a project that lasts 90 days, the total cost of human resources is 16 650€. We have estimated the cost of the hours of each of the roles using a study of the IT sector in Galicia [28].

6.2.2 Software Resources

All the mentioned tools in Chapter 4 are open source or have a free student license, so these will not add any cost to our project.

6.2.3 Hardware Resources

- Personal laptop:
  - Model: Dell Inspiron 14 5000 Series 2-in-1 -5482.
  - CPU: Intel® Core™ i7-8565U.
  - Hard disk: 512GB Solid State Drive (SSD).
  - RAM: 16GB.
  - OS: Windows 10 Home (64 bits).
  - Graphic card: Intel® UHD 620.

The cost of the hardware is calculated as the time of use of this type of resources during the project in relation to their lifespan. For a laptop, we consider that its average life is four years. So reviewing the cost of ours that is 1 098.99€ and dividing it by 4, we obtain the cost

per year (274.75€). As our project lasts 3 months, that is, a quarter of a year, the cost of our hardware will be 68.69€.

6.2.4 Estimated Total Cost

Once we have analysed all the types of resources and their associated costs, we can calculate the total project cost, that is:

Resources	**Cost**
Human	16,650€
Software	0€
Hardware	68.69€
TOTAL	16,718.69€

Table 6.2: Estimated cost of total project resources.

6.3 Project Follow-up

Our project has followed the plan of the project, except for the implementation of the *taxPerformance* function, which has required considerably more hours than expected due to its complexity.

On the other hand, not all the tasks have exactly fulfilled the planned duration, but we have compensated the hours. The graphical interface has taken a few more days since it has not only been implementing, but also learning how to use the tool itself (JavaFX). However, thanks to having been doing an internship, the process of implementing the model and part of the service was faster than scheduled.

The need to spend more hours on the project is also an extra cost, since we have more hours of work by the three team members and also more hours of use of our hardware.

While with respect to hardware, increasing the duration of the project does not increase the cost even by 4€, with respect to human resources, it will mean a cost growth of more than 1,000€.

Table 6.3 reflects the impact of those deviations on the initial time and cost planning for this project.

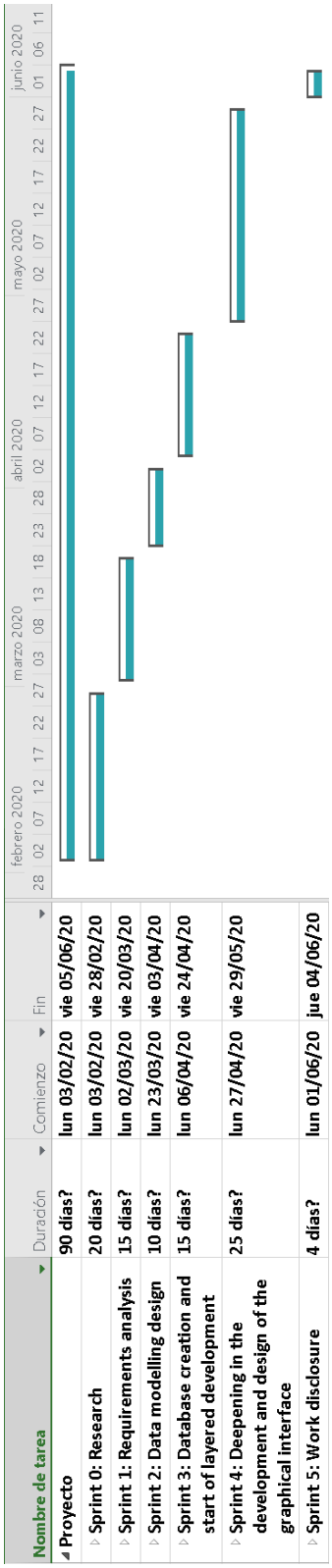
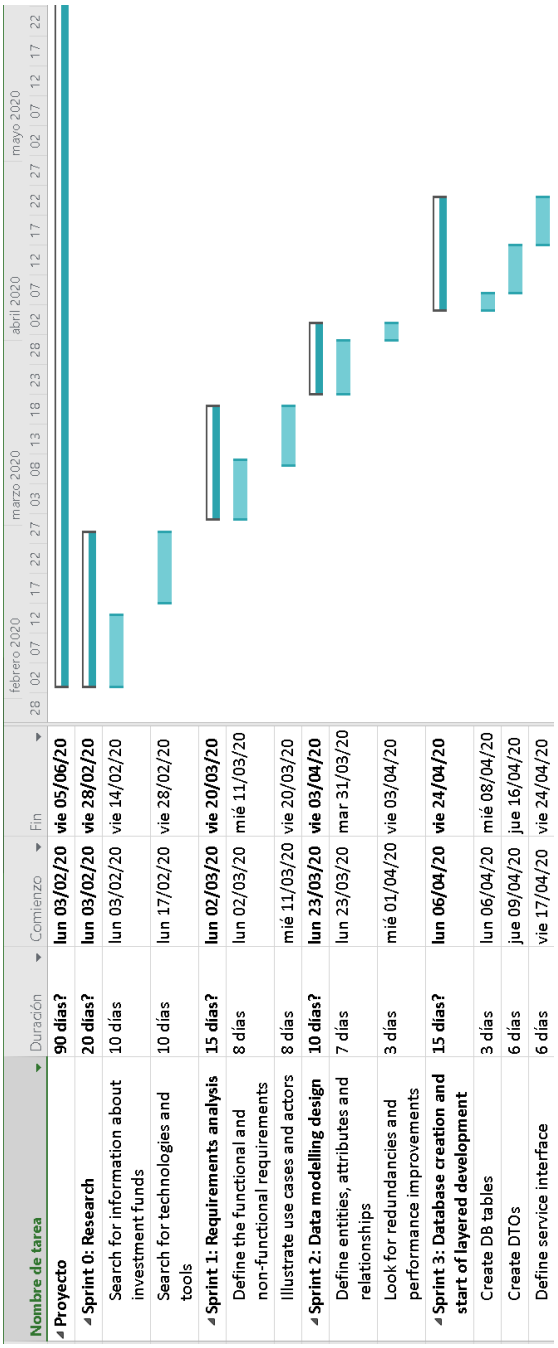
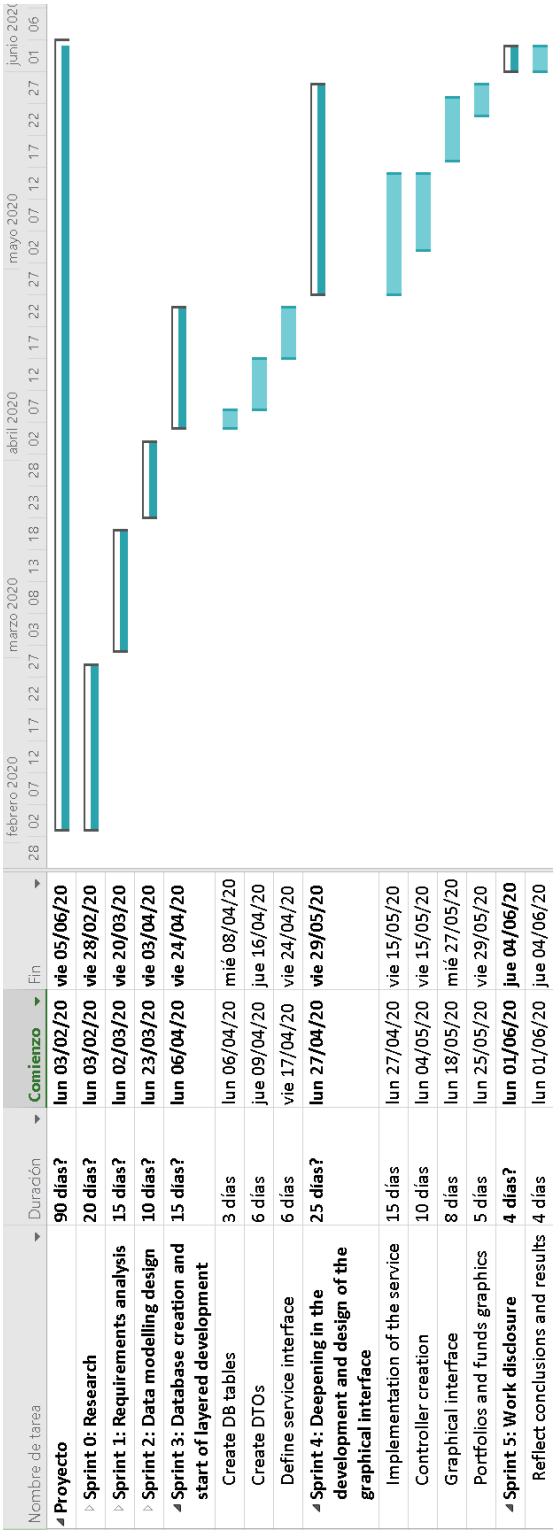


Figure 6.1: Gantt diagram with the 6 Sprints.



Nombre de tarea	Duración	Comienzo	Fin
Proyecto	**90 días?**	**lun 03/02/20**	**vie 05/06/20**
Sprint 0: Research	**20 días?**	**lun 03/02/20**	**vie 28/02/20**
Search for information about investment funds	10 días	lun 03/02/20	vie 14/02/20
Search for technologies and tools	10 días	lun 17/02/20	vie 28/02/20
Sprint 1: Requirements analysis	**15 días?**	**lun 02/03/20**	**vie 20/03/20**
Define the functional and non-functional requirements	8 días	lun 02/03/20	mié 11/03/20
Illustrate use cases and actors	8 días	mié 11/03/20	vie 20/03/20
Sprint 2: Data modelling design	**10 días?**	**lun 23/03/20**	**vie 03/04/20**
Define entities, attributes and relationships	7 días	lun 23/03/20	mar 31/03/20
Look for redundancies and performance improvements	3 días	mié 01/04/20	vie 03/04/20
Sprint 3: Database creation and start of layered development	**15 días?**	**lun 06/04/20**	**vie 24/04/20**
Create DB tables	3 días	lun 06/04/20	mié 08/04/20
Create DTOs	6 días	jue 09/04/20	jue 16/04/20
Define service interface	6 días	vie 17/04/20	vie 24/04/20

Figure 6.2: Gantt diagram with the first 3 Sprints and their specific tasks.



Nombre de tarea	Duración	Comienzo	Fin
Proyecto	**90 días?**	**lun 03/02/20**	**vie 05/06/20**
Sprint 0: Research	**20 días?**	**lun 03/02/20**	**vie 28/02/20**
Sprint 1: Requirements analysis	**15 días?**	**lun 02/03/20**	**vie 20/03/20**
Sprint 2: Data modelling design	**10 días?**	**lun 23/03/20**	**vie 03/04/20**
Sprint 3: Database creation and start of layered development	**15 días?**	**lun 06/04/20**	**vie 24/04/20**
Create DB tables	3 días	lun 06/04/20	mié 08/04/20
Create DTOs	6 días	jue 09/04/20	jue 16/04/20
Define service interface	6 días	vie 17/04/20	vie 24/04/20
Sprint 4: Deepening in the development and design of the graphical interface	**25 días?**	**lun 27/04/20**	**vie 29/05/20**
Implementation of the service	15 días	lun 27/04/20	vie 15/05/20
Controller creation	10 días	lun 04/05/20	vie 15/05/20
Graphical interface	8 días	lun 18/05/20	mié 27/05/20
Portfolios and funds graphics	5 días	lun 25/05/20	vie 29/05/20
Sprint 5: Work disclosure	**4 días?**	**lun 01/06/20**	**jue 04/06/20**
Reflect conclusions and results	4 días	lun 01/06/20	jue 04/06/20

Figure 6.3: Gantt diagram with the last 3 Sprints and their specific tasks.

	Planned	Follow-up
Start Date	03/02/2020	03/02/2020
Finish Date	05/06/2020	15/06/2020
Duration	90 days	96 days
Work	630 hours	672 hours
Cost	16,718.69€	17,832.26€

Table 6.3: Comparison between forecast and follow-up.

Chapter 7

Requirements Analysis

Global analysis of application requirements is a process of conceptualization and formulation of the concepts that specifically involves. It is a fundamental part of the application development process, most of the defects found in the the final product originate from the requirements analysis phase and they are also the most expensive to repair. This chapter corresponds to Sprint 1, in the same way as Chapters 2 and 4 correspond to Sprint 0.

7.1 Functional Requirements

Functional requirements are product features or functions that developers must implement to enable users to accomplish their tasks. Our project has the followings:

- **Add fund**. The system must allow adding a new fund to the database.
- **Find funds by keywords**. Funds which contain certain words in their names are returned.
- **Add portfolio**. The system must allow adding a new portfolio to the database.
- **Add fund to portfolio**. We can add a new fund or an existing one to a portfolio.
- **Remove fund from portfolio**. Our system allows to remove funds that do not have transactions for that portfolio.
- **Add transaction**. Add a new transaction for a specific fund and portfolio, it can have as source cash or another fund with its matching portfolio.
- **Add NAV**. The system allows to add net asset values to a specific fund.
- **Import NAVs**. We can import net asset values from an URL.

- **Import currencies**. The system allows to import currencies from a .csv file obtained from the European Central Bank.
- **View all funds in a portfolio**. We can see all the funds associated with a portfolio.
- **Display transactions for a specific fund and portfolio**. We show all transactions whether they are deposits or refunds of a specific fund and portfolio.
- **View NAVs of a fund**. Show all the Net Asset Values of a fund.
- **Display graphs and calculated metrics for a fund or portfolio**. The user will be able to see the ratios explained in the Chapter 2 or some representative graphs.

7.2 Non-Functional Requirements

Non-Functional Requirements (NFR) specify the quality attribute of a software system. They judge the software system based on Responsiveness, Usability, Security, Portability and other non-functional standards that are critical to the success of the software system. The NFRs of the application are:

- **Usability**. It is the ease at which the users operate the system and make productive use of it.
- **Integrity**. The degree to which the data maintained by the software system are accurate, authentic, and without corruption.
- **Reliability**. The extent to which the software system consistently performs the specified functions without failure.

7.3 Use Cases

In this section we detail the different use cases for each of the functional requirements.

Name	Add a new fund.
Description	Add a new fund to the database.
Trigger	The client requests to add a fund.
Actors	User.
Preconditions	-
Flow	1. The user requests to add a fund. 2. The system provides a dialog to enter the data. 3. The user enters the data. 4. The user confirms the data. 5. The system adds the fund to the database.
Alternative flow	4.1 The user cancels the operation.
Postconditions	The fund will belong to the database.
Exceptions	The ISIN already exists or has an incorrect format.

Table 7.1: Add a fund.

Name	Find funds by keywords.
Description	We show the funds which have keywords in their name.
Trigger	Write in a search engine the word or letters that will contain the name of the funds.
Actors	User.
Preconditions	-
Flow	1. The user types what he wants in the search engine. 2. The system shows us the funds that contain keywords in all the portfolios that are.
Alternative flow	-
Postconditions	-
Exceptions	-

Table 7.2: Find funds by keywords.

Name	Add a new portfolio.
Description	Add a new portfolio to the database.
Trigger	The client requests to add a portfolio.
Actors	User.
Preconditions	-
Flow	1. The user requests to add a portfolio. 2. The system provides a dialog where to enter the data. 3. The user enters the portfolio data. 4. The user confirms what has been entered. 5. The system adds the portfolio to the database.
Alternative flow	4.1 The user cancels the operation.
Postconditions	The portfolio must belong to the database.
Exceptions	The name of the portfolio is not null or repeated.

Table 7.3: Add a portfolio.

Name	Add an existing fund to a portfolio.
Description	An existing fund will be added to a portfolio.
Trigger	The client requests to add an existing fund to a portfolio.
Actors	User.
Preconditions	The fund and portfolio exist in the database. Have a selected portfolio.
Flow	1. The user requests to add an existing fund to a portfolio. 2. The system provides a dialog that shows the funds that exist to be added to the portfolio in which we are. 3. The user confirms the operation. 4. The system adds the fund to the portfolio in the database.
Alternative flow	3.1 The user cancels the operation.
Postconditions	The fund is related to the portfolio in the database.
Exceptions	The fund is already added to that portfolio.

Table 7.4: Add fund to portfolio.

Name	Remove a fund from a portfolio.
Description	Remove a fund from a portfolio.
Trigger	The client requests to remove a fund from a portfolio.
Actors	User.
Preconditions	The fund and portfolio exist in the database. Have a portfolio and fund selected.
Flow	1. The user requests to remove a fund from a portfolio. 2. The system provides a confirmation dialog to remove the selected fund. 3. The user confirms the operation. 4. The system removes the fund from the portfolio in the database.
Alternative flow	3.1 The user cancels the operation.
Postconditions	The fund is no longer related to that portfolio.
Exceptions	The fund did not belong in that portfolio.

Table 7.5: Remove fund from portfolio.

Name	Add a transaction.
Description	Add a transaction between funds in the same portfolio.
Trigger	The client requests to add a transaction.
Actors	User.
Preconditions	The portfolio and funds exist in the database.
Flow	1. The user requests to add a transaction. 2. The system provides a dialog to enter the data of the transaction. 3. The user enters the data. 4. The user confirms the data is correct. 5. The system adds the transaction to the database.
Alternative flow	4.1 The user cancels the operation.
Postconditions	The transaction must belong to the database.
Exceptions	Target date is greater than today or source date greater than target date. Source fund does not have enough money to transfer the indicated amount.

Table 7.6: Add a transaction.

Name	Add a new NAV.
Description	Add a NAV to the selected fund.
Trigger	The client requests to add a NAV.
Actors	User.
Preconditions	The fund to which we want to add the NAV exists in the database. Fund must be selected.
Flow	1. The user requests to add a NAV. 2. The system provides a dialog to enter the data of the NAV. 3. The user enters the data. 4. The user confirms the data is correct. 5. The system adds the NAV to the database.
Alternative flow	4.1 The user cancels the operation.
Postconditions	The NAV must belong to the database.
Exceptions	Date is greater than today. The value of the NAV is negative.

Table 7.7: Add a NAV.

Name	Import NAVs.
Description	Import a set of Net Asset Values next to the fund they belong to from a URL or only NAVs if the fund already exists.
Trigger	The client requests to import NAVs.
Actors	User.
Preconditions	-
Flow	1. The user requests to import NAVs. 2. The system provides the option to open the browser with the website from where we import the funds. 3. The system displays a dialog where you can copy the URL of the background you want. 4. The user confirms the operation. 5. The system adds the NAVs and, maybe, the fund to the database.
Alternative flow	4.1 The user cancels the operation.
Postconditions	The NAVs and the fund must belong to the database.
Exceptions	Those net asset values already exist for that fund.

Table 7.8: Import NAVs.

Name	Import currencies.
Description	Import the change of euro to other currencies from a .csv file for a specific date.
Trigger	The client requests to import currencies.
Actors	User.
Preconditions	-
Flow	1. The user requests to import currencies. 2. The system opens a window that allows to select the file to import. 3. The user selects the file. 4. The user confirms the operation. 5. The system adds the currencies to the database.
Alternative flow	4.1 The user cancels the operation.
Postconditions	The currencies must belong to the database.
Exceptions	-

Table 7.9: Import currencies.

Name	View all funds in a portfolio.
Description	Show all funds added to the selected portfolio with their details.
Trigger	The client requests to view funds from that portfolio.
Actors	User.
Preconditions	The portfolio must exist in the database. Have a portfolio selected.
Flow	1. The user requests to view funds from portfolio. 2. The system displays the funds that belong to the portfolio.
Alternative flow	-
Postconditions	The system displays the funds that belong to the portfolio.
Exceptions	-

Table 7.10: View all funds in a portfolio.

Name	Display transactions for a specific fund and portfolio.
Description	Show transactions with their detail for a fund and portfolio.
Trigger	The client requests to view transactions.
Actors	User.
Preconditions	The portfolio and fund must exist in the database. Have a fund selected.
Flow	1. The user requests to view transactions from portfolio and fund. 2. The system displays the transactions that belong to fund and portfolio.
Alternative flow	-
Postconditions	The system displays the transactions that belong to the fund and portfolio.
Exceptions	-

Table 7.11: Display transactions from a specific fund and portfolio.

Name	View NAVs of a fund with its customize graphic.
Description	Show NAVs of the selected fund and the graph that represents them for the selected range.
Trigger	The client requests to view NAVs.
Actors	User.
Preconditions	The portfolio and fund must exist in the database. Have a fund selected.
Flow	1. The user requests to view NAVs from fund. 2. The system displays the NAVs that belong to selected fund and its graph.
Alternative flow	-
Postconditions	The system displays the NAVs and its graphic.
Exceptions	-

Table 7.12: View NAVs of selected fund.

Name	Display graphs and calculated profitabilities and metrics.
Description	Show the ratios, the returns and the graph of a fund or the graphs of a portfolio.
Trigger	The client requests to view graphs or/and metrics from fund or portfolio.
Actors	User.
Preconditions	The portfolio and/or fund must exist in the database. Have a portfolio or fund selected.
Flow	1. The user requests to view metrics, returns and graph from fund. 2. The system displays the ratios, profitabilities and graphics that belong to selected fund.
Alternative flow	1.1 The user requests to view graphs from portfolio. 2.1 The system displays graphics that belong to selected portfolio.
Postconditions	The system displays metrics, profitabilities and/or graphics.
Exceptions	-

Table 7.13: Display graphs and calculated profitabilities and metrics.

Chapter 8

Design

System design is the process of designing the elements of a system such as the architecture and the data that goes through that system. Of course, this chapter corresponds to Sprint 2.

8.1 Architecture

The architecture follows the Model View Controller (MVC) design pattern, which specifies that an application consist of a data model, presentation information, and control information. The pattern requires that each of these be separated into different objects.

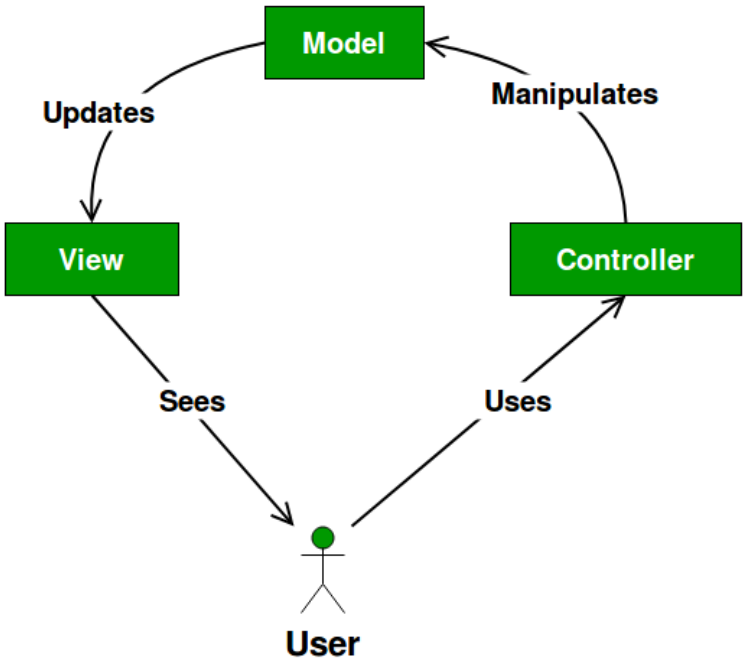


Figure 8.1: MVC Design Pattern.

- The **Model** contains only the pure application data, it contains no logic description on how to present the data to a user.
- The **View** presents the model's data to the user. The view knows how to access the model's data, but it does not know what this data means or what the user can do to manipulate it.
- The **Controller** exists between the view and the model. It listens to events triggered by the view (or another external source) and executes the appropriate reaction to these events. In most cases, the reaction is to call a method on the model.

8.2 Database Design

As we can see in the following figure, we have five tables that represent the 6 classes of our project and in addition one table resulting from the N-to-N relationship between funds and portfolios.

1. *Portfolio.* It is a collection of investment funds. It consists of a name and an id that will be the primary key.
2. *Fund.* A fund can belong to one, several or no funds. Each fund has a name, an ISIN, deposit and refund fees, and an associated currency. Also, it has an auto-generated id as primary key.
3. *Transaction.* A transaction is associated with at least one fund and one portfolio (as destination, with a certain date). In addition, you can have another association in case that the transaction operates between two funds (in this case we would need another date), i.e., the amount invested is not coming from cash. Finally, of course, we must indicate the amount we want to invest, by default it will be in euros, regardless of the currencies of both funds. Lastly, they will also have an id as the primary key.
4. *NAV.* A net asset value has as attributes a date and a value, associated with a fund. In the same way as the previous entities, the primary key is an id.
5. *Currency.* For the currency entity we have the attribute: target, that is the exchange of euro to certain currency called. The primary key continues to be an auto-generated id.
6. *Spot.* In the same way that we have the NAV entity with respect to the funds, we have this entity associated with the currencies, its attributes are spot (the change from euro to that currency) and date, since for each date the spot varies. Like the previous 5 classes, the primary key is an auto-generated id.

All the ids are autogenerated, including *Fund* class where we could use the candidate key ISIN (unique identifier), defined as a *Varchar* that must be unique and not null. It was decided that it would be the autogenerated id like the others for performance reasons, since the primary key (PK) will be repeated in each of the large number of NAVs.

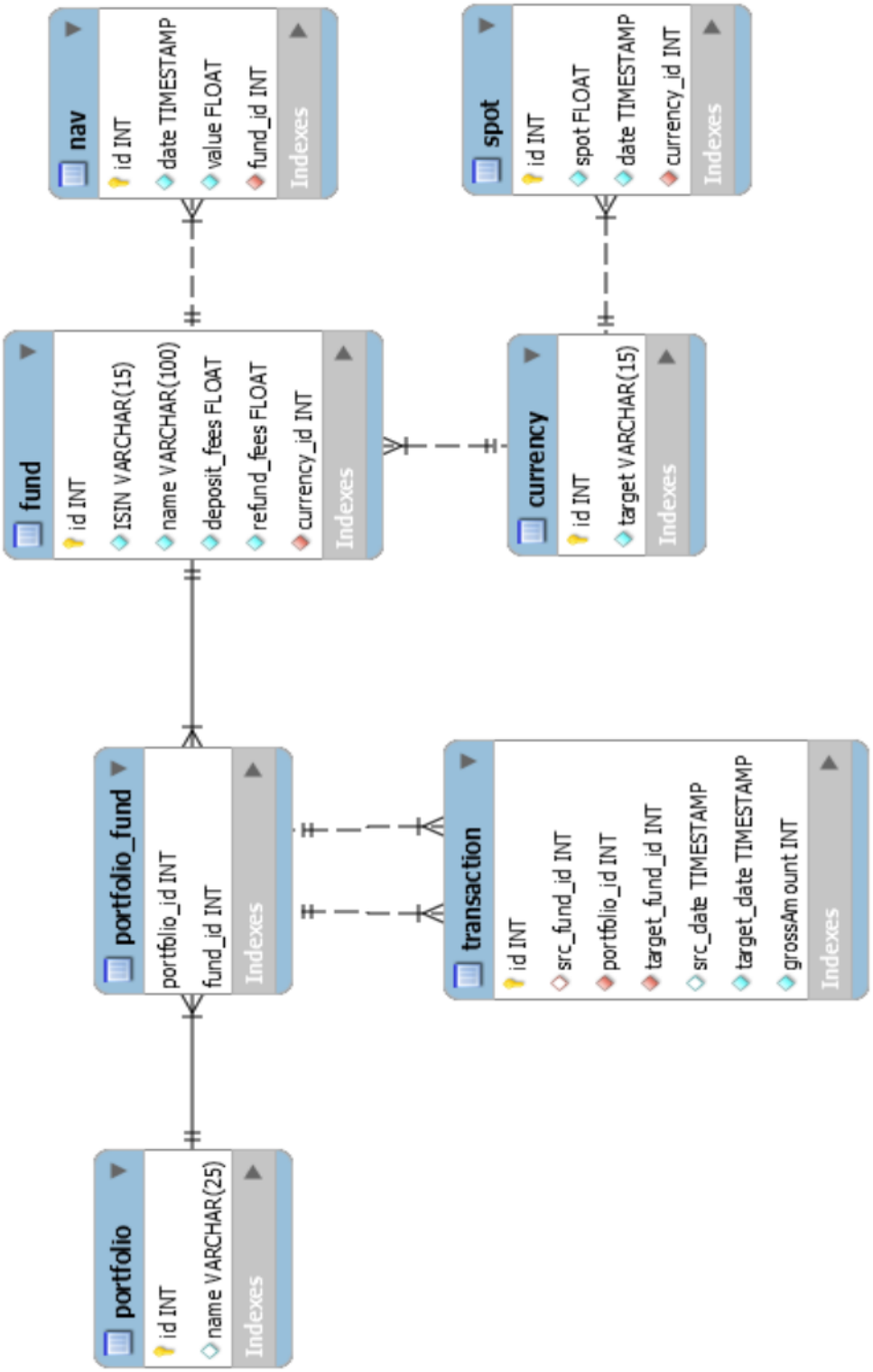


Figure 8.2: Database design.

Chapter 9
Development

In this chapter, we will explain the most relevant aspects of the entire development phase, which corresponds to Sprints 4 and 5

9.1 Database Implementation

To implement the database using Hibernate, we must first model the Java classes corresponding to the tables of the relational model, as shown in Fig. 9.1.

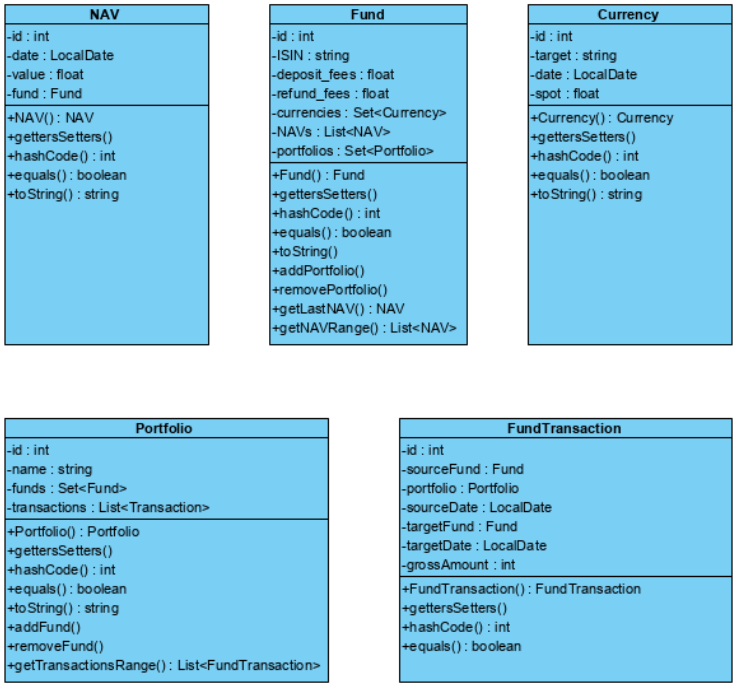


Figure 9.1: Model class diagram.

All *id* attributes, which correspond to the primary key of each class, are created by the database with the generation strategy *GenerationType.IDENTITY*, that means autoincrement. This is a concept supplied by MySQL, that other database management softwares like Oracle don't provide.

In order to map the relationships (1:N, N:1 and N:N), Hibernate needs us to use its annotations to know what kind of relationship there is, if there will be a cascade effect in any action, how to load the data, etc.

Below, we indicate the different annotations we have used and their definition:

- *@Id.* It is indicated that it is the primary key of the table.
- *@GeneratedValue.* You can define the identifier generation strategy thanks to this annotation.
- *@Column.* The column used for a property mapping is defined using this annotation.
- *@ManyToOne.* Many-to-one associations are declared at the property level with this annotation. In conjunction with @ManyToOne, we use @JoinColumn which is used to indicate the join column, of course.
- *@OneToMany* [29]. One-to-many associations are declared at the property level with this annotation. In this case, the attributes are lists of the convenient object, such as the Net Asset Values in the *Fund* class. With the attribute *mappedBy* [30] refers to the property name of the association on the owner side.

  We also use *cascade* [31] attribute to indicate transitive persistence and cascading of operations in Hibernate. Continuing with the example of the NAVs, when we delete a fund, all its Net Asset Values will be deleted, so we must indicate *CascadeType.REMOVE*.

  We have the ability to either eagerly or lazily fetch associated entities. The fetch parameter can be set to *FetchType.LAZY* or *FetchType.EAGER* [32]. *EAGER* will try to use an outer join select to retrieve the associated object, while *LAZY* will only trigger a select when the associated object is accessed for the first time.
- *@ManyToMany* [33]. A many-to-many association is defined logically using the @ManyToMany annotation. You also have to describe the association table and the join conditions using the @JoinTable annotation. The *JoinTable* defines a foreign key to the source object primary key (*joinColumns*) [30] and a foreign key to the target object primary key (*inverseJoinColumns*). Normally the primary key of the *JoinTable* is the combination of both foreign keys.

  It is important to know that the attribute that we associate with this annotation must be a set, not a list. Otherwise, it gives problems of constraints violation.

- *@Fetch*. In addition to the *FetchType*, we will also talk about the *FetchMode*. Which *FetchMode* to use depends heavily on the application, environment and typical usage. In the case of NAVs, we will use *FetchMode.SUBSELECT* which will provide us with greater performance.

- *@OrderBy*. To order lists in memory, we use this annotation that takes into parameter a list of comma separated properties and order the collection accordingly.

The correct configuration of the *hibernate.cfg.xml* file is important. There we define the classes that will be mapped, the url that we need to connect to the DB, the user and the password. We will also indicate that console queries are displayed to facilitate the recognition and fix of errors during implementation.

We have created a class called **HibernateUtil** where the methods where the session is created (*buildSessionFactory*) and closed (*shutdown*) are implemented. When we make the login call, in the event that they are not yet built, the tables will be created.

9.2 Service Implementation

In the service, we will implement the logic of our application. The methods necessary to add, search or delete each object, depending on their needs (for example, you cannot delete transactions because we need to have a record of all of them), will be found in this interface with its corresponding implementation.

In the next diagram of Fig. 9.2 we have an interface for each table in the database and in this way see the operations related to each one, but the reality is that all these methods are united in the same interface.

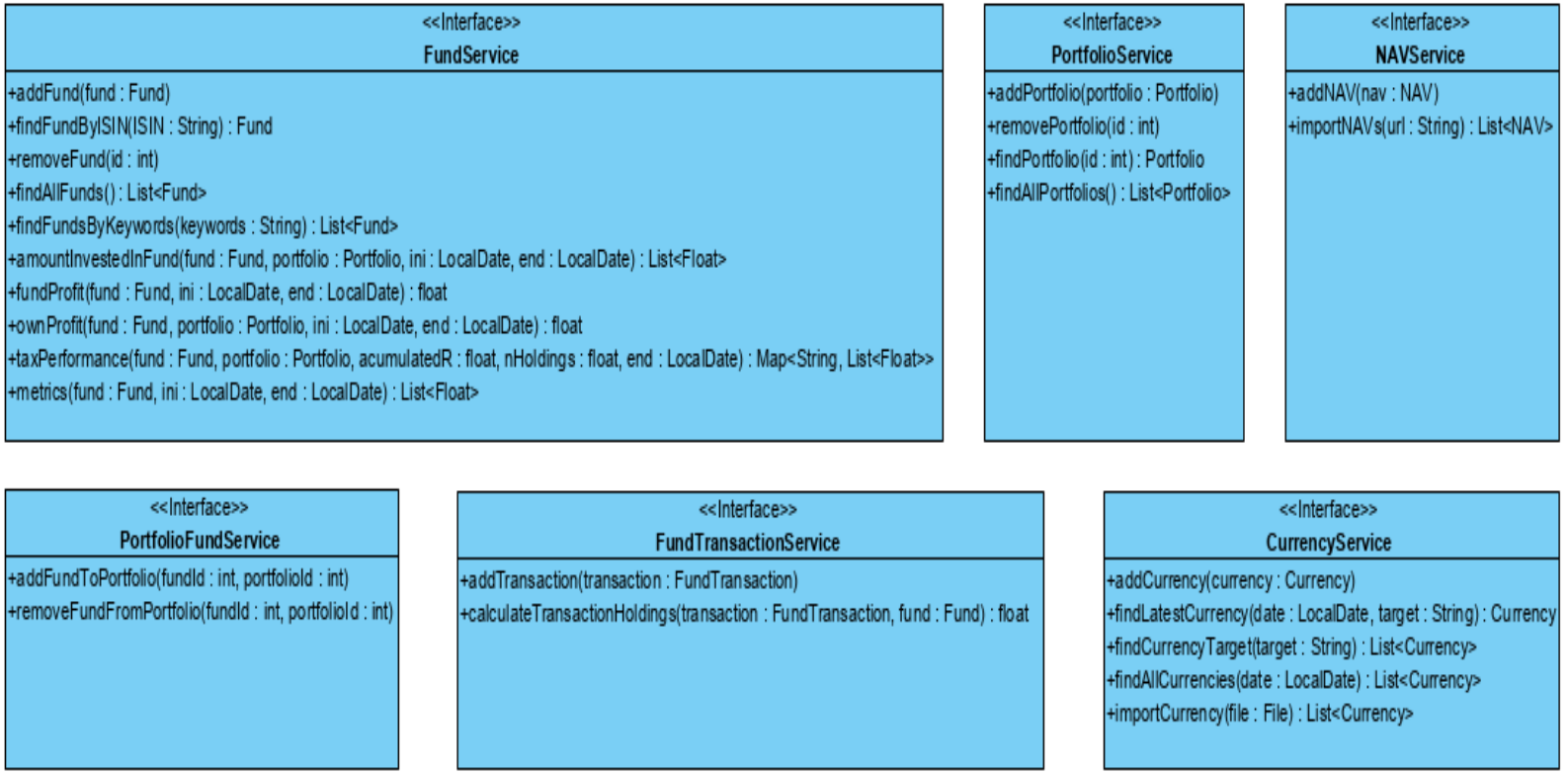


Figure 9.2: Service diagram.

In the event that these methods do not have the expected behaviour, they may throw one of these exceptions or both, depending on the operation, as follows,

- *InstanceNotFoundException* [34]. As its own name says, it occurs when the desired element is not found (either when we search, modify or delete).
- *InputValidationException.* In the event that the data entered is not correct, this exception will be returned.

To ensure that the entered data meets the requirements, we have created a class exclusively dedicated to validate the input parameters, it is called **PropertyValidator**.

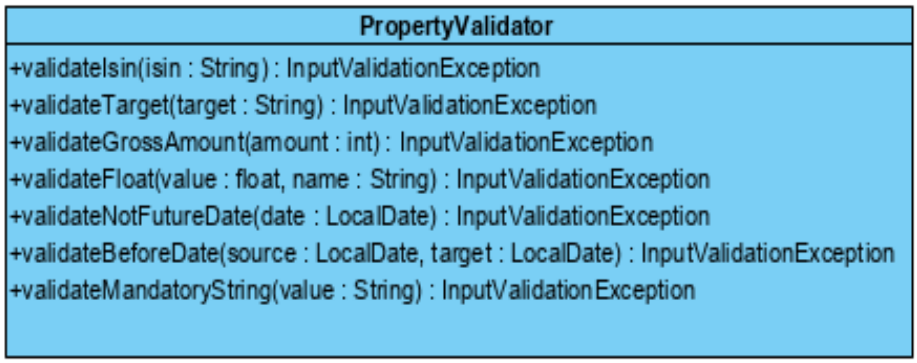


Figure 9.3: PropertyValidator class diagram.

In the service, we have a method for each class that calls the *PropertyValidator* operations based on their needs:

- *validateFund.* We have to validate that the ISIN has the correct format (two letters and 10 numbers below), that the name is a non-empty string, that the commissions are a float greater than 0 and at the same time validate that the objects it contains (currency and NAVs) are also correct.
- *validateNAV.* For the Net Asset Values, we check that the value is a float greater than 0 and that the date is not later than the current day.
- *validateTransaction.* In this case we will check if there is *sourceDate* and if so, that it is not later than the *targetDate* or today. We must also validate *targetDate* in the same way. And, finally, we will check that the *grossAmount* is an integer greater than 0 and in the case of having a *sourceFund*, that this fund has enough shares to make that refund.

- *validateCurrency*. Validates that target is a string containing three characters, that the date is not later than the current day and the value is a float greater than 0.
- *validatePortfolio*. Portfolio only has one parameter to check and it is the *name*, so there is no method as such but we call *validateMandatoryString*.

Now, we will proceed to explain each method in depth:

- **Fund**:
  - *addFund*. We add a fund to the database. Each fund will have a currency associated with it and at first it will not be added to any portfolio, this will be a later step.
  - *findFundByISIN*. ISIN means unique identifier so this method will return, if it exists, the fund with the indicated ISIN.
  - *removeFund*. We indicate the fund id and, if there is one, we will delete it with its corresponding NAVs.
  - *findAllFunds*. This method returns all the existing funds in the database.
  - *findFundsByKeywords*. It obtains all the funds that contain in its name the string that we pass to it as a parameter.
  - *amountInvestedInFund*. This method returns a list of floats, which include the amount invested for that fund in that portfolio, total holdings, refunds and deposits. We do this calculation for an indicated period or for all fund transactions.
  - *fundProfit*. Using the formula already explained in the Chapter 2, we calculate the performance of the fund for the period we want.
  - *ownProfit*. We do not only calculate the performance of the fund, but also the performance that we will have depending on how and when we have made the deposits and refunds for a specific fund and a portfolio.
  - *taxPerformance*. As with all returns and metrics, they are already explained above. Although in the case of this performance, it goes far beyond calculations since we have to go back both for the transaction history and for the net asset values' one. We will have to see if the investment comes from cash or if it is from another fund, if it is the last option, we must trace how those shares came to that fund.

    Tracking all the funds our transactions have gone through is not an easy task, so before implementing in our IDE, we need to previously write a pseudocode.

Algorithm TaxPerformance

```
1: Input: fund, portfolio, acumulatedR, nHoldings, endDate
2: refunds ← add pre-endDate refunds
3: Discard deposits corresponding to refunds (FIFO)
4: interestTransactions<n> ← Select from transactions not discarded so that the sum
   of the holdings of these transactions of interest equals nHoldings.
5: taxPerf ← new ArrayList
6: splitHold ← new ArrayList
7:    for all n in interestTransactions do
8:        if n.sourceFund == null then
9:            currentR ← NAV(endDate) / NAV(n.targetDate)
10:           taxPerf ← add currentR × acumulatedR
11:           splitHold ← add n.targetHoldings
12:       else
13:           currentR ← NAV(endDate) / NAV(n.targetDate)
14:           (taxPerfSource, splitHoldSource) ← taxPerformance(n.sourceFund, n.portfolio,
               currentR × acumulatedR, n.sourceHoldings, n.sourceDate)
15:           taxPerf ← add taxPerfSource
16:           totalHoldSource ← Sum of splitHoldSource
17:           for all elements in splitHoldSource do
18:               element = element × nHoldings / totalHoldSource
19:           end for
20:           splitHold ← add splitHoldSource
21:       end if
22:   end for
23: return taxPerf, splitHold
```

First with respect to discarding the purchases that correspond to the sales, if the sold units that we have left are less than the participations of the transaction, it will be the first transaction that we will be added to those that interest us but remaining only with the participations resulting from subtracting the sold.

The same will happen with the last transaction, in the event that the shares that interest us (*nHoldings*) is less than the number of shares in the transaction, we will add this transaction but only with the remaining holdings.

Once we have the list with all the transactions that interest us added, we will proceed to go through it. If the transaction that we are processing does not have a source fund, that is, it comes from cash, we will calculate the profitability and the shares associated with that transaction, and add both values to each array of results.

On the contrary, if the transaction comes from another fund, we will calculate the profitability for this transaction and we will make a recursive call to this same function. Next, with the results of the recursive call we will add the returns to its

corresponding array of results. And finally, since the shares that the recursive call returns to us correspond to the fund of the recursive call, not the current one, we will do the calculation to obtain the portion of shares that correspond to our fund and we will add it to the list of results of participations.

At last, after going through all the transactions, we will return the two result arrays with returns and participations.

- *metrics*. This method brings together all the metrics calculated for that fund.

- **Portfolio**:
  - *addPortfolio*. With this operation we add a portfolio to the database.
  - *removePortfolio*. We pass the *id* as a parameter and delete the corresponding portfolio.
  - *findPortfolio*. In the same way, we will have the *id* as input and in the event that there is a fund that identifies with it, we will return it.
  - *findPortfolio*. We will provide all funds saved in the database.

- **Portfolio_Fund**:
  - *addFundToPortfolio*. We pass a fund id and a portfolio id. And in the event that both exist, we add that fund to that portfolio.
  - *removeFundFromPortfolio*. In this case, if the fund id corresponds to a fund that belongs to the portfolio that we also indicate, we will delete the relation between them from the database; unless there are transactions for that fund and that portfolio.

- **FundTransaction**:
  - *addTransaction*. We add the transaction that we pass as a parameter to the database. Although this transaction must meet the requirements that we already indicated when explaining *PropertyValidator* class.
  - *calculateTransactionHoldings*. We pass as input the transaction and the fund of which we want to know the shares (that is, it can be source or target fund), so we will return the calculation of the holdings.

- **Currency**:
  - *addCurrency*. We add a currency to the database.
  - *findLatestCurrency*. Having as input a currency and a date, we return the currency with a date equal to or less than that indicated.

  - *findCurrencyTarget*. This function returns all the currencies of a specific target.
  - *findAllCurrencies*. We will provide all the currencies stored in the database.
  - *importCurrency*. We will import all the currencies that a .csv file contains that we pass as a parameter.
  - *addCurrency*. We add a currency to the database.
  - *findLatestCurrency*. Having as input a currency and a date, we return the currency with a date equal to or less than that indicated.
  - *findCurrencyTarget*. This function returns all the currencies of a specific target.
  - *findAllCurrencies*. We will provide all the currencies stored in the database.
  - *importCurrency*. We will import all the currencies that a .csv [35] file contains that we pass as a parameter.

- **NAV**:
  - *addNAV*. We add a NAV to the fund that is indicated in the NAV class itself.
  - *importNAVs*. From the url that we pass as a parameter, we add the NAVs and the fund to which they are associated to the database.

9.3 GUI

In this section we will describe the interface design and how we have implemented it. Thanks to the JavaFX tool (see Section 4.6) we have been able to implement all the methods defined in the service in our desktop application.

To start we will have to create the window that will contain all the elements of our interface, this object is called *JavaFX Stage* [36]. Inside a *Stage* you can insert a *JavaFX Scene* [37] which represents the content displayed inside a window (*Stage*).

9.3.1 Main Screen

We will describe the interface starting with its main screen and the elements that persist through the screens. As our model of portfolios and funds is very similar to a tree structure, being the portfolios the parent nodes and each fund a leaf, we will allow with a *JavaFX TreeeView* [38] see at a glance the relationships between them, as we saw in the Fig. 9.4.

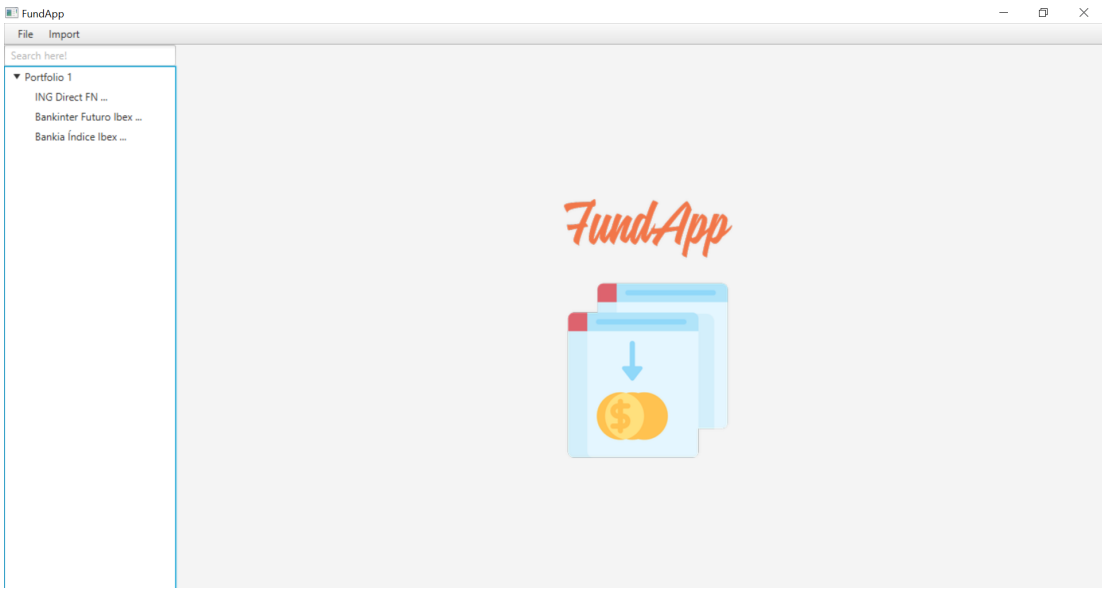


Figure 9.4: Main Screen.

Also, we will include *JavaFX MenuBar* [39] provides JavaFX applications with a visual drop down menu similar to that most desktop applications have at the top of their application window. We will have a *Menu* called *File* (see Fig. 9.5) where we will have the possibility of adding a portfolio (see Fig. 9.6) or a fund (see Fig. 9.7).

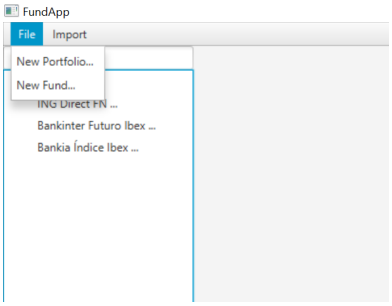


Figure 9.5: *File* Menu displayed.

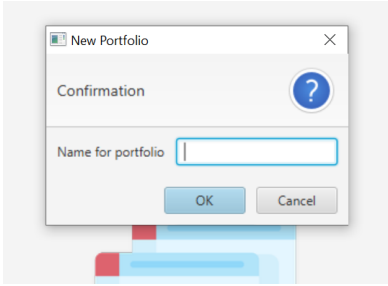


Figure 9.6: Dialogue that we get to add a new portfolio.

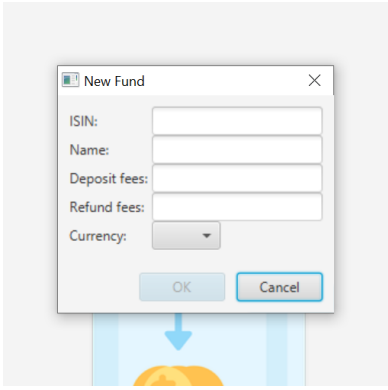


Figure 9.7: Dialogue that we get to add a new fund.

We will also have another Menu (see Fig. 9.8) that will include the two types of import that our service offers: import currencies (see Fig. 9.9) and / or import NAVs (see Fig. 9.11) with their corresponding funds from the chosen website (see Fig. 9.10).

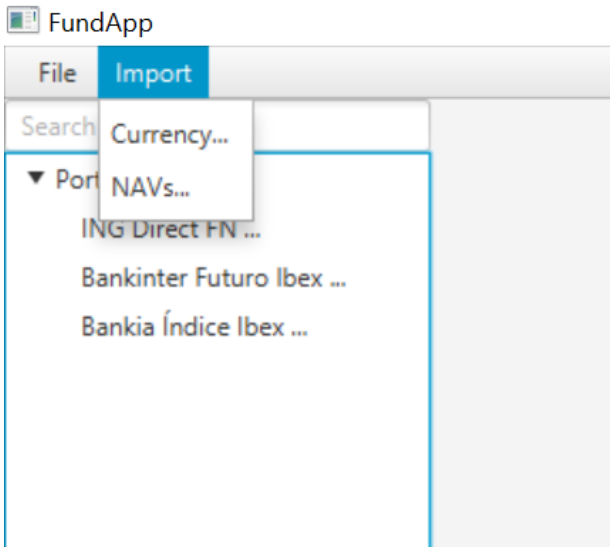


Figure 9.8: *Import* Menu displayed.

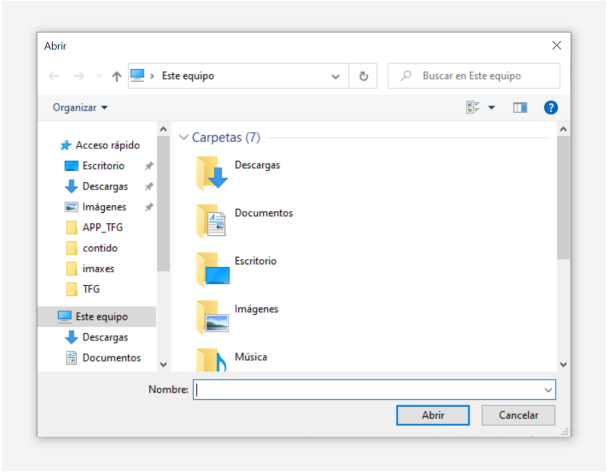


Figure 9.9: Dialogue that allows us to select a file to import currency.

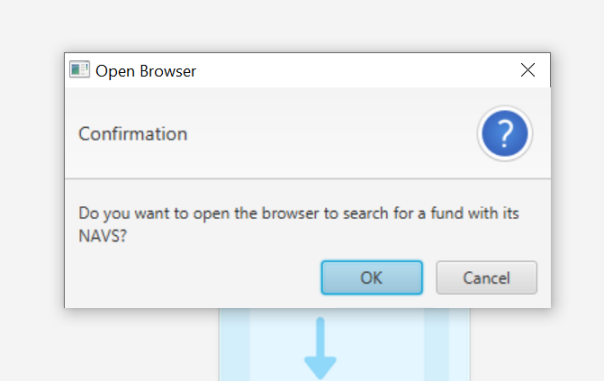


Figure 9.10: Dialogue that allows us to open the website where we choose the NAVs to import.

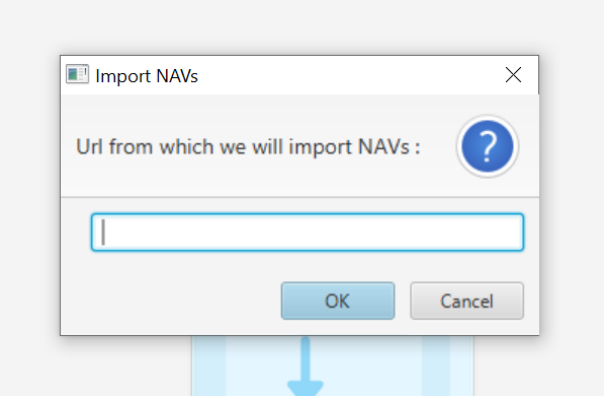


Figure 9.11: Dialogue where we enter the URL with the NAVs.

Finally, we will have a *TextField* that, depending on what we write, will filter the *TreeView* (see Fig. 9.12 and Fig. 9.13) only showing the funds that include the indicated string in their name.

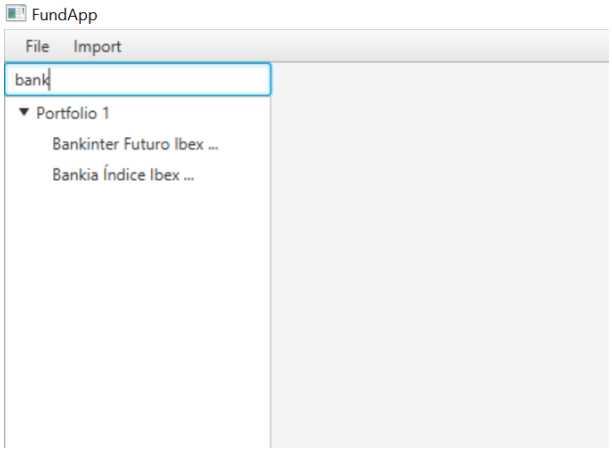


Figure 9.12: Example of using the searcher returning some funds.

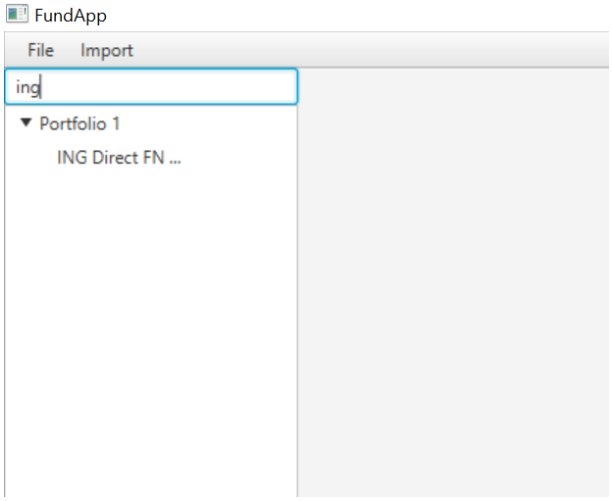


Figure 9.13: Example of using the searcher returning a fund.

9.3.2 Portfolio Screen

When we click on a portfolio, we have a new deck of possibilities and all of them pertain to the selected portfolio. In a *JavaFX TableView* [40] we will represent the funds that belong to that portfolio. This table has the following columns: ISIN, name, deposit fees, refund fees, last NAV and currency.

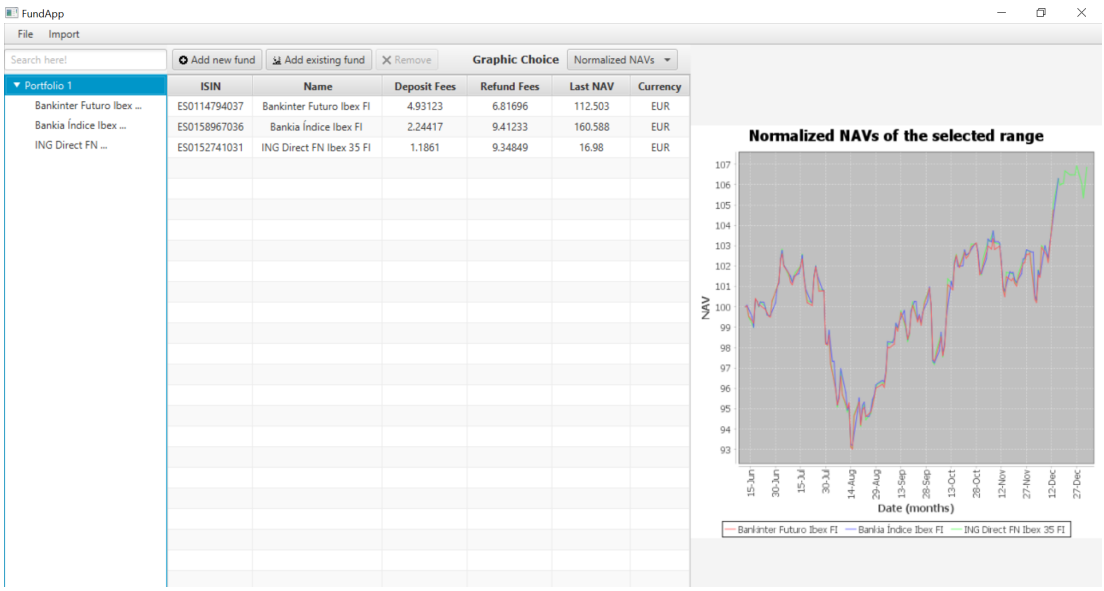


Figure 9.14: Screen shown when clicking on portfolio.

At the top of the table we have a *JavaFX ToolBar* [41] (horizontal or vertical bar containing buttons), which contains several buttons (*JavaFX Button*) [42] and a *JavaFX ChoiceBox* [43]. First of all, we have two buttons:

- *Add New Fund* (see Fig. 9.15). With this button we will add to this portfolio a fund that is not yet in the database, so when we accept we will add it to the DB and to the portfolio. We introduce the data thanks to a *JavaFX Dialog* [44] and the answer will be to be added or show an *Alert* with the appropriate error.
- *Add Existing Fund* (see Fig. 9.16). In this case the funds already existing in the DB will be shown in a *JavaFX ListView* [45] in a new *Dialog*. And once we choose one, when accepting it will be added to the selected portfolio.

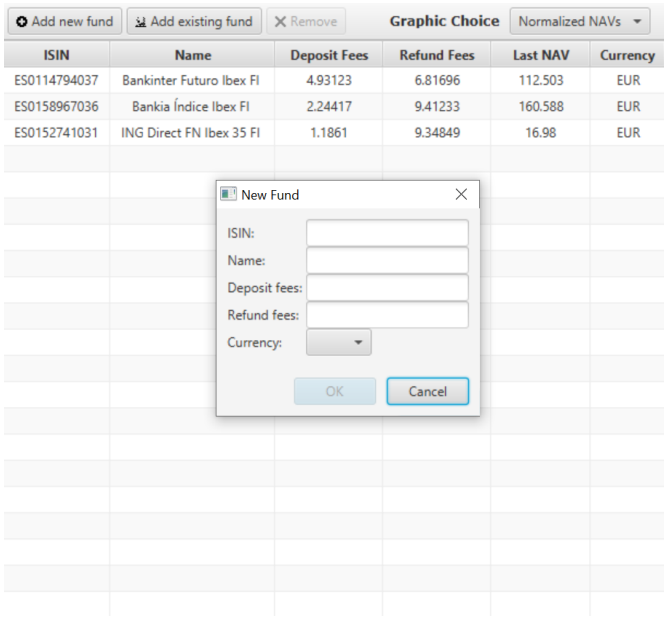


Figure 9.15: Dialogue that we get to add a new fund.

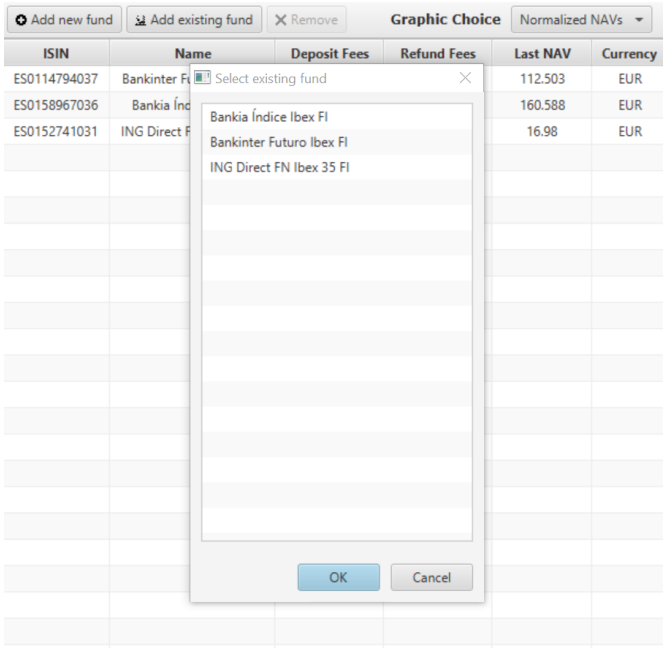


Figure 9.16: Dialogue where we choose among the existing funds which we add.

To remove a fund, we will click on the one we want to remove and the button that allows this action will be enabled (see Fig. 9.17). After pressing this button, a confirmation dialog will appear (see Fig. 9.18) and we will decide if it is permanently deleted.

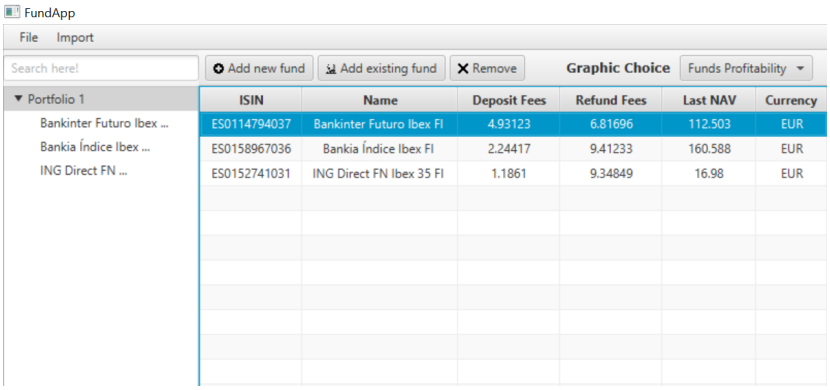


Figure 9.17: Pressing on a fund activates the button to delete it.

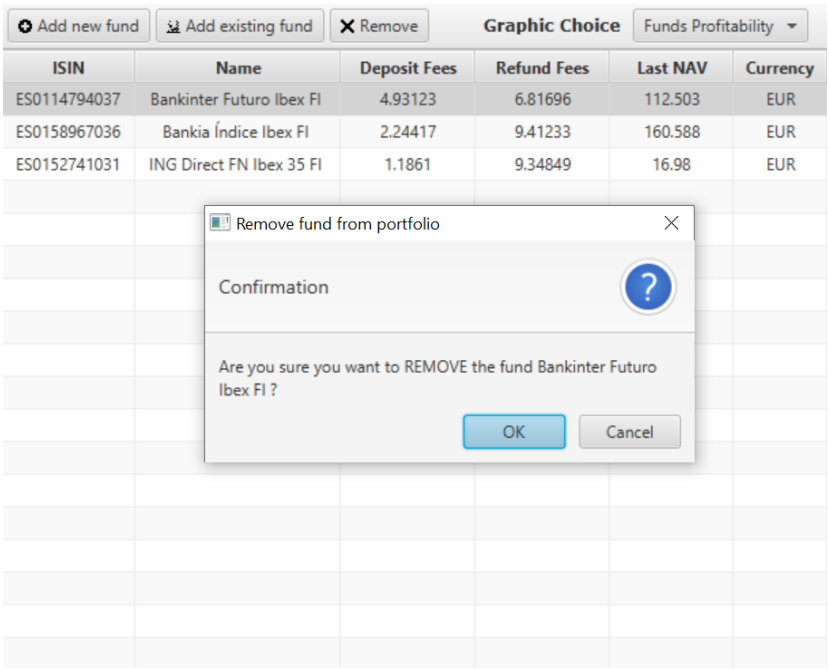


Figure 9.18: Confirmation of deletion of the selected fund.

Finally, we have a *ChoiceBox* where we will choose which graph we want to see. We have three options and any of the three refer to the content of that portfolio:

- *Normalized NAVs.* In this case we can compare the net asset values of the funds in our portfolio, since they have been normalized. This is the graph that comes out by default, as we see in the previous image, Fig. 9.14.

- *Invested In* (see Fig. 9.19). In this graph we will show the amount invested in each fund with respect to the total amount of the portfolio using a pie chart.

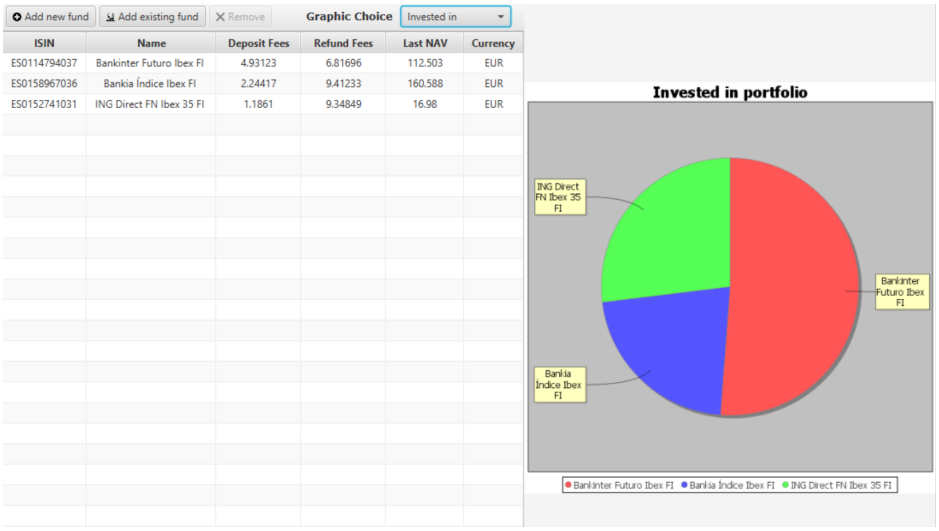


Figure 9.19: Example of the graph representing investments.

- *Funds Profitability* (see Fig. 9.20). Thanks to this graph we can compare the returns of the funds in the same period of time (their last year with registered values).

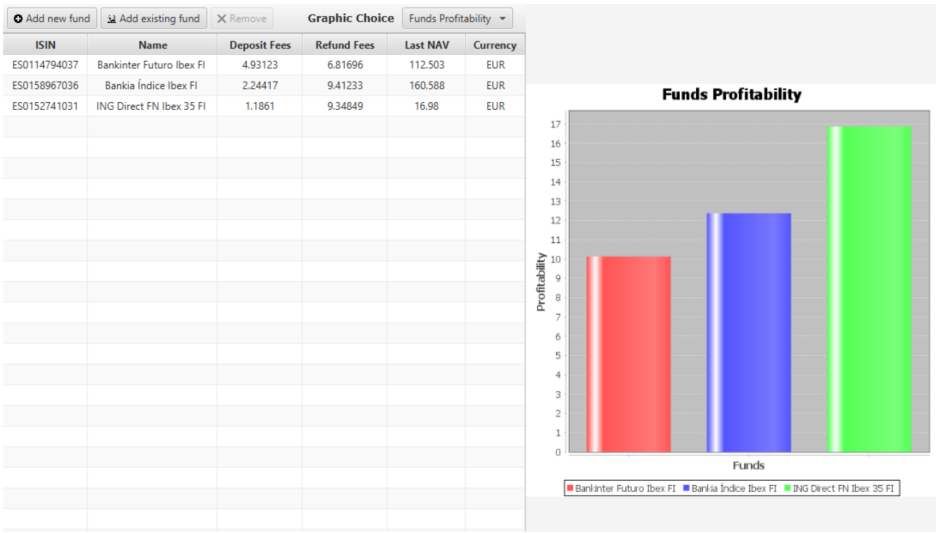


Figure 9.20: Example of Funds Profitability graphic.

9.3.3 Fund Screen

When we click on a fund we will find a *JavaFX TabPane* [46] that is a container control which can contain multiple tabs (sections) internally, which can be displayed by clicking on the tab. In this case we have three tabs:

- *NAV* (see Fig. 9.21). When we click on the tab called NAV, we will see a table that shows us the value of the NAV, the date and the difference between that NAV and the previous one (see Fig. 9.22).

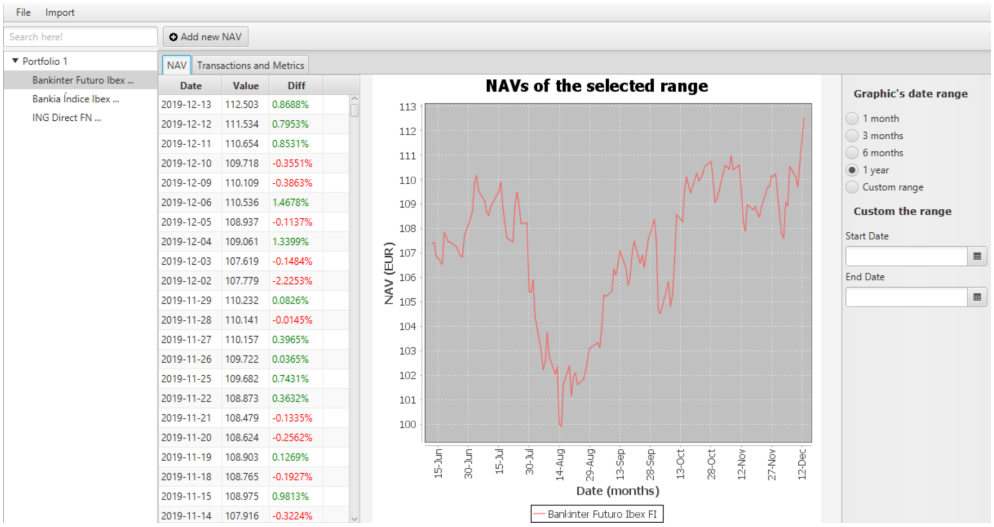


Figure 9.21: *NAV* tab.

Date	Value	Diff
2019-12-13	112.503	0.8688%
2019-12-12	111.534	0.7953%
2019-12-11	110.654	0.8531%
2019-12-10	109.718	-0.3551%
2019-12-09	110.109	-0.3863%
2019-12-06	110.536	1.4678%
2019-12-05	108.937	-0.1137%
2019-12-04	109.061	1.3399%
2019-12-03	107.619	-0.1484%
2019-12-02	107.779	-2.2253%
2019-11-29	110.232	0.0826%
2019-11-28	110.141	-0.0145%
2019-11-27	110.157	0.3965%
2019-11-26	109.722	0.0365%
2019-11-25	109.682	0.7431%
2019-11-22	108.873	0.3632%
2019-11-21	108.479	-0.1335%
2019-11-20	108.624	-0.2562%
2019-11-19	108.903	0.1269%
2019-11-18	108.765	-0.1927%
2019-11-15	108.975	0.9813%
2019-11-14	107.916	-0.3224%

Figure 9.22: NAVs table.

In addition to the *importNAVs* option previously explained, we can also add a NAV manually thanks to this dialog (see Fig. 9.23) that will open when you press the *Add new NAV* button. As you can see the dialogue is simple, we only have to enter the value and the desired date.

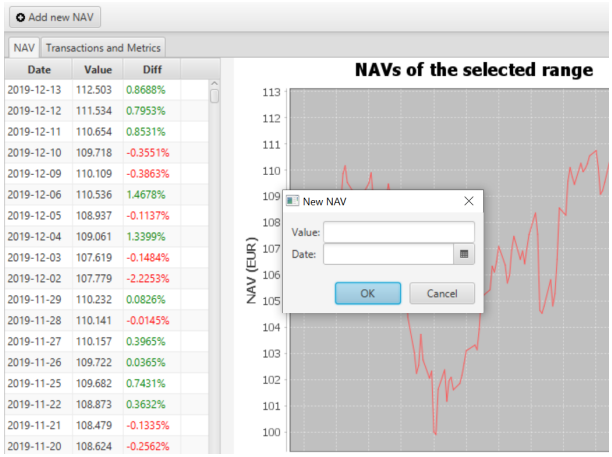


Figure 9.23: Dialog that allows adding a new NAV.

We will also display a representative graphic of NAVs over a certain period of time. To the left of the table we have a vertical *ToolBar* with a *JavaFX ToggleGroup* [47] which is a set of *RadioButton* that allows at most one to be selected at any time (see Fig. 9.24). Each button gives the option of a date range: one month, three months, six months, one year or at last the option of customize the range. So under the *ToggleGroup*, we have two *JavaFX DatePicker* [48] that will allow us to select the start and the end date of the range.

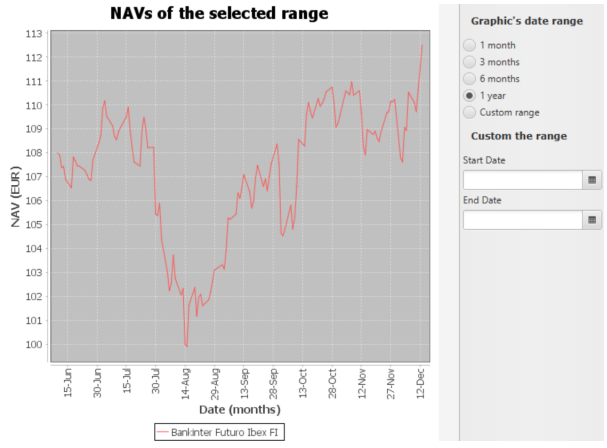


Figure 9.24: Graph and its corresponding *ToolBar*.

- **Transactions and Metrics** (see Fig. 9.25): In this tab we will show a *TableView* that contains all the transactions related to that fund, that is, when it is source and when it is target. The columns that make up this table are: Source Fund, Source Date, Source holdings, Source NAV, Target Fund, Target Date, Target Holdings, Target NAV and Gross Amount.

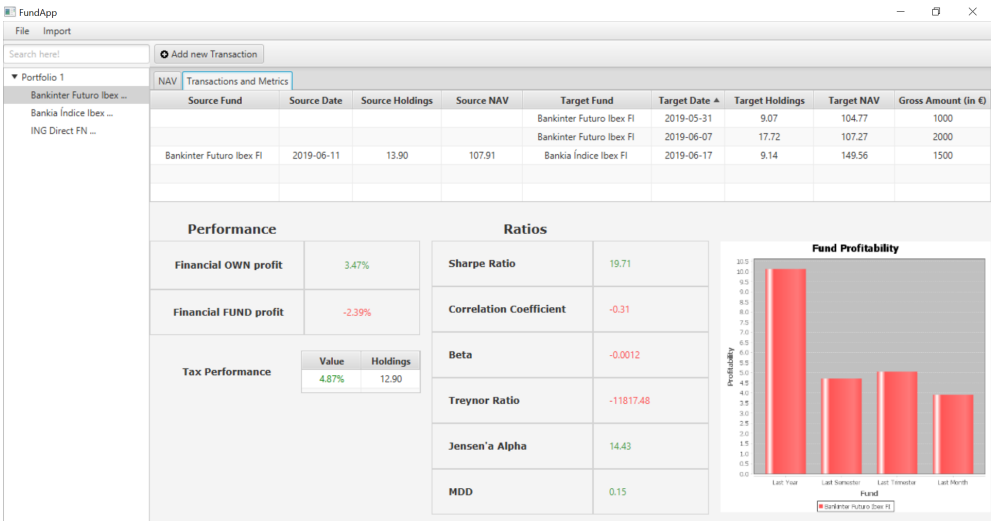


Figure 9.25: *Transactions and Metrics* tab.

Over the table we will have a button that allows us to add a new transaction. Pressing it opens a dialog (see Fig. 9.26) where we will indicate the portfolio, the target fund and the source's one if it exists, their corresponding dates and the gross amount. Since we need to have the history of all the transactions to calculate the tax return, we will not allow to eliminate any transaction. This will be the tab selected by default.

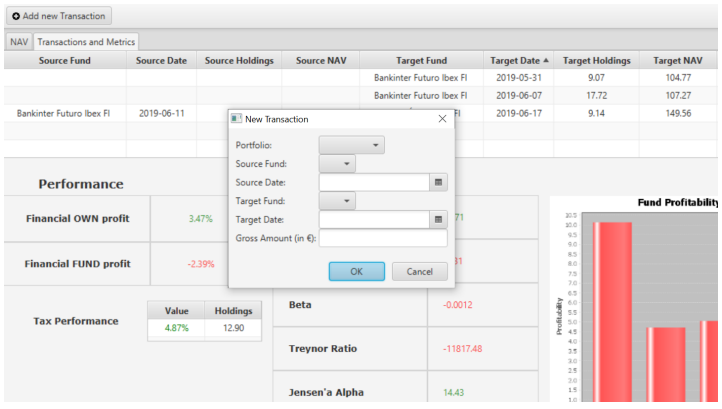


Figure 9.26: Dialog shown to add a new transaction.

Besides, we will visualize the three calculated returns: that of the fund, that calculated from the transactions of our portfolio and the fiscal one (see Fig. 9.27).

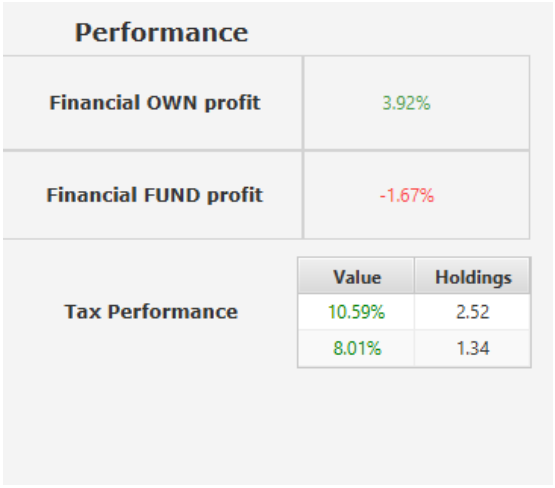


Figure 9.27: Performance results.

On the other hand we have the value of all the ratios previously explained: Sharpe Ratio, Correlation Coefficient, Beta, Treynor Ratio, Jensen's Alpha and Maximum Drawdown (see Fig. 9.28).

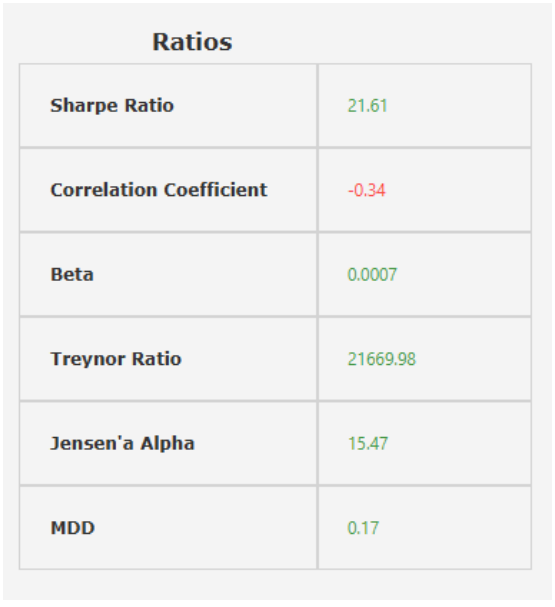


Figure 9.28: Ratios calculations.

Finally, we will have a graph that compares the fund's returns in four different periods (see Fig. 9.29).

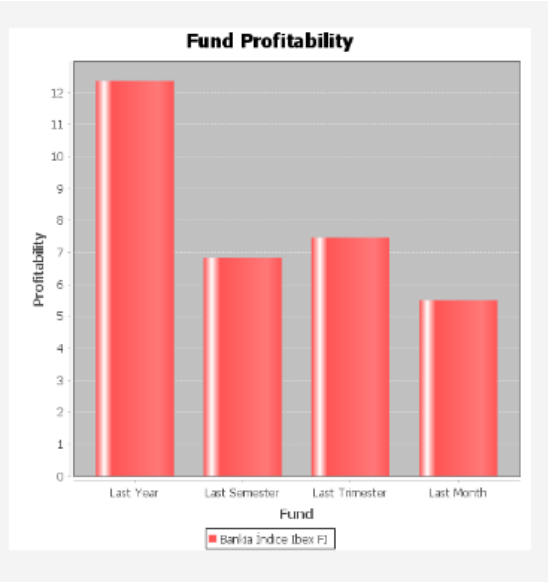


Figure 9.29: Fund Profitability graph.

Chapter 10

Tests

This chapter explains the tests carried out on the different aspects of our application.

10.1 Model Layer Tests

We must check that all the methods provided by the service work correctly. For this purpose we have created *ServiceTest* class which uses JUnit.

Fifty tests are implemented in this class that automatically and exhaustively check the correct operation of each of the methods created in the service.

It has been very important for these tests to check both the successful and the failed cases. We must verify that when an entered data does not meet the requirements that we have specified, it throws the relevant error and that if we are trying to access something that does not exist in the database, it informs us of an *InstanceNotFoundException*.

These tests have evolved as the code has done. That is, when a change has occurred in a function, such as when we realized that before adding a transaction where source fund is different from null, we should check that this fund has sufficient amount of investment. Therefore, we had to adapt the test already done and create more tests to check if that condition is not met.

10.2 Functionality Testing

These tests focus on determining if the interface correctly performs the functions for which it has been implemented.

In this case, they have been carried out manually, intensively testing all the components in order to verify that they react as they are supposed to, and if they do not, detect and correct them.

Having previously proven that all methods of the service were working correctly, it has

saved us a lot of work at this stage. Likewise, in some cases we had escaped showing an alert informing why that operation was failing and thanks to these tests, all the exceptions thrown have their corresponding alert.

10.3 Usability Testing

With these tests we have verified that the interface is intuitive and easy to use.

These tests were made possible with the help of the Product Owner, since the person who has implemented the interface has a different point of view and it is more difficult to see the deficiencies in terms of intuitiveness and ease. Besides, as we have already explained, one of the Product Owner's responsibilities is to understand the user's needs, making him the most appropriate to participate in these tests.

With these tests we have perfected the interface, adjusting the table and graph sizes for a better visualization, modifying the positions of the buttons to make them more intuitive, etc.

10.4 Graphs Tests

In the same way that it happens in the interface, we do not have an automated way to test the obtained graphics. We have also needed the help of our Product Owner to verify that these graphics are displayed as expected.

To ensure correct operation, we have reviewed the operations carried out carefully and verified that what the graphs shown agree with the results of these operations.

Chapter 11

Conclusions and Future Work

In this last chapter we will explain the analysis, evaluation and conclusions obtained from our project and how the application we have implemented could continue growing.

11.1 Conclusions

Firstly, it should be noted that the final product fulfils all the functionalities that we had as goal. It is an application that allows us to have one (or several) portfolio with the funds that we wish to add and that provides us detailed and tailored information of the funds in which we have decided to invest and taking into account how and when we have done so. The application does the currency exchange between funds itself and takes into account all the history of NAVs that we can import (or add individually if we prefer) and on all the registered transactions.

It is also basic in our application, obtaining returns and ratios that will help us with our future investments. On the one hand, we calculate the fund's return and the most useful ratios to be able to make decisions comparing return-risk, what is expected with what is actually obtained, etc. On the other hand, we will return the profitability calculated from the movements that have been made in a certain fund, not only taking into account the values of the NAVs in a certain period, which is what is calculated in all applications, websites, banks... And as the most outstanding functionality we find fiscal profitability. The reason is threefold: First due to its difficulty for implementation and the elegant solution; second, because this functionality cannot be found on any other platform; and finally, it is a metric that can make the difference between the money that we could earn thanks to this calculation.

The importance of the functionalities of our project lies not only in its great usefulness, but that no application, bank or website provides most of them. So we have developed an application with a great future projection since the information we offer will be appreciated by any investor.

All this in a simple and intuitive interface that abstracts the complexity of the system for the user. Always allowing the users to form their portfolios with the funds they want and above all to select and customize the information that will be displayed.

The realization of this project has not been a challenge only in terms of development but has also required a great immersion in the world of investment funds. Discover the operation of a fund, the meaning of the capital gains, what are the ratios and returns and how they will help us in our investment, etc. Knowing all this information is not only necessary to be able to use and understand the application, but is of interest in how we could increase our capital.

11.2 Future Work

The application is ready to be tested in a real environment and meets the objectives set, but it could continue growing.

Right now we are allowing the user to import the currencies that he/she considers necessary, but to improve it we could make the application automatically import the currencies for the dates in which we have transactions, which is when we will use this change.

Another future line of work could also be that the application evaluating the returns and the metrics calculated for our funds, which would recommend us which investments it considers most appropriate.

This application was intended to be a desktop application, but it would also be very interesting if it were available for mobile phones.

Finally, another possibility would be to open the application to other types of investments such as the stock market or the currency market. In this way the user profile would be extended to other types of investors, not only those of investment funds.

Appendices

List of Acronyms

API *Application Programming Interface.*

CPU *Central Processing Unit.*

DAG *Directed Acyclic Graph.*

DB *Database.*

DTO *Data Transfer Object.*

GUI *Graphical User Interface.*

HTML *HyperText Markup Language.*

IDE *Integrated Development Environment.*

ISIN *International Securities Identification Number.*

IT *Information Technology.*

MDD *Maximum Drawdown.*

NAV *Net Asset Value.*

OS *Operating System.*

PK *Primary Key.*

POJO *Plain Old Java Objects.*

POM *Project Object Model.*

RAM *Random Access Memory.*

RDBMS *Relational Database Management System.*

RUP *Rational Unified Process.*

SSD *Solid State Drive.*

UI *User Interface.*

UML *Unified Modeling Language.*

XML *eXtensible Markup Language.*

Bibliography

[1] I. F. Hugo Gutiérrez, "El pánico hunde unas bolsas en caída libre: el ibex se desploma un 14,06su historia," 2020. [Online]. Available: https://elpais.com/economia/2020-03-12/las-bolsas-sufren-en-la-apertura-y-el-ibex-cae-mas-de-un-5.html

[2] S. Matamoros, "Mapfre am: el coronavirus y el crudo refuerzan un escenario de tipos bajos o negativos," 2020. [Online]. Available: https://www.expansion.com/mercados/2020/03/10/5e676de0468aeb0b2c8b465a.html

[3] J. C. Bogle, *Bogle on Mutual Funds*, wiley investment classic ed. Wiley, 2015.

[4] Wikipedia, "International securities identification number," 2020. [Online]. Available: https://en.wikipedia.org/wiki/International_Securities_Identification_Number

[5] J. Chen, "Net asset value – nav," 2020. [Online]. Available: https://www.investopedia.com/terms/n/nav.asp

[6] J. Alarcón, "Operativa en los fondos de inversión," 2017. [Online]. Available: https://inbestia.com/analisis/operativa-en-los-fondos-de-inversion

[7] E. C. Bank, "Euro foreign exchange reference rates," 2020. [Online]. Available: https://www.ecb.europa.eu/stats/policy_and_exchange_rates/euro_reference_exchange_rates/html/index.en.html

[8] REALIA, "Plusvalía fiscal," 2019. [Online]. Available: https://www.realia.es/plusvalia-fiscal

[9] BBVA, "Tipos de fondos de inversión," 2020. [Online]. Available: https://www.bbva.es/finanzas-vistazo/ef/fondos-inversion/tipos-de-fondos-de-inversion.html

[10] C. Banton, "What is a good annual return for a mutual fund?" 2019. [Online]. Available: https://www.investopedia.com/ask/answers/050415/what-good-annual-return-mutual-fund.asp

[11] Selfbank, "¿qué es el ratio de sharpe?" 2020. [Online]. Available: https://www.selfbank.es/centro-de-ayuda/fondos-de-inversion/que-es-el-ratio-de-sharpe

[12] W. Kenton, "Beta," 2020. [Online]. Available: https://www.investopedia.com/terms/b/beta.asp

[13] C. F. Institute, "What is the treynor ratio?" 2020. [Online]. Available: https://corporatefinanceinstitute.com/resources/knowledge/finance/treynor-ratio/

[14] J. Chen, "Jensen's measure," 2019. [Online]. Available: https://www.investopedia.com/terms/j/jensensmeasure.asp

[15] A. P. Ucha, "Coeficiente de correlación lineal," 2020. [Online]. Available: https://economipedia.com/definiciones/coeficiente-de-correlacion-lineal.html

[16] Robeco, "The formula: Maximum drawdown," 2018. [Online]. Available: https://www.robeco.com/es/vision-del-mercado/2018/04/the-formula-maximum-drawdown.html

[17] "Morning star," 2020. [Online]. Available: https://www.morningstar.com/

[18] "inverco," 2020. [Online]. Available: http://www.inverco.es/

[19] "Fonditus," 2020. [Online]. Available: https://play.google.com/store/apps/details?id=com.sierralion.fonditus&hl=es

[20] "Finizens," 2020. [Online]. Available: https://finizens.com/

[21] MySQL, "Mysql: General infromation," 2020. [Online]. Available: https://dev.mysql.com/doc/refman/8.0/en/introduction.html

[22] developer.com, "Hibernate basics for java persistence," 2020. [Online]. Available: https://www.developer.com/java/other/article.php/3559931/Hibernate-Basics.htm

[23] tutorialspoint, "Javafx - overview," 2020. [Online]. Available: https://www.tutorialspoint.com/javafx/javafx_overview.htm

[24] EDUCBA, "Difference between java swing vs java fx," 2020. [Online]. Available: https://www.educba.com/java-swing-vs-java-fx/

[25] javaTpoint, "Jsoup api," 2018. [Online]. Available: https://www.javatpoint.com/jsoup-api

[26] Scrum.org, "What is scrum?" 2020. [Online]. Available: https://www.scrum.org/resources/what-is-scrum

[27] Deloitte, "Scrum: roles y responsabilidades," 2020. [Online]. Available: https://www2.deloitte.com/es/es/pages/technology/articles/roles-y-responsabilidades-scrum.html

[28] V. Boullosa, "Guia salarial sector ti galicia 2015-2016," 2016. [Online]. Available: https://es.scribd.com/document/288511179/Guia-Salarial-Sector-TI-Galicia-2015-2016

[29] Baeldung, "Hibernate one to many," 2020. [Online]. Available: https://www.baeldung.com/hibernate-one-to-many

[30] S. Srivastava, "Difference between @joincolumn and mappedby," 2020. [Online]. Available: https://www.baeldung.com/jpa-joincolumn-vs-mappedby

[31] L. Gupta, "Hibernate jpa cascade types," 2018. [Online]. Available: https://howtodoinjava.com/hibernate/hibernate-jpa-cascade-types/

[32] T. Janssen, "Class instancenotfoundexception," 2020. [Online]. Available: https://thorben-janssen.com/entity-mappings-introduction-jpa-fetchtypes/

[33] Z. H. Baeldung, "Hibernate many to many," 2018. [Online]. Available: https://www.baeldung.com/hibernate-many-to-many

[34] J. P. S. E. 7, "Class instancenotfoundexception," 2018. [Online]. Available: https://docs.oracle.com/javase/7/docs/api/javax/management/InstanceNotFoundException.html

[35] baeldung, "Reading a csv file into an array," 2019. [Online]. Available: https://www.baeldung.com/java-csv-file-array

[36] J. Jenkov, "Javafx stage," 2018. [Online]. Available: http://tutorials.jenkov.com/javafx/stage.html

[37] ——, "Javafx scene," 2018. [Online]. Available: http://tutorials.jenkov.com/javafx/scene.html

[38] ——, "Javafx treeview," 2019. [Online]. Available: http://tutorials.jenkov.com/javafx/treeview.html

[39] ——, "Javafx menubar," 2018. [Online]. Available: http://tutorials.jenkov.com/javafx/menubar.html

[40] ——, "Javafx tableview," 2020. [Online]. Available: http://tutorials.jenkov.com/javafx/tableview.html

[41] ——, "Javafx toolbar," 2018. [Online]. Available: http://tutorials.jenkov.com/javafx/toolbar.html

[42] ——, "Javafx button," 2019. [Online]. Available: http://tutorials.jenkov.com/javafx/button.html

[43] ——, "Javafx choicebox," 2016. [Online]. Available: http://tutorials.jenkov.com/javafx/choicebox.html

[44] P. Saya, "Javafx dialog," 2015. [Online]. Available: https://examples.javacodegeeks.com/desktop-java/javafx/dialog-javafx/javafx-dialog-example/

[45] J. Jenkov, "Javafx listview," 2016. [Online]. Available: http://tutorials.jenkov.com/javafx/listview.html

[46] ——, "Javafx tabpane," 2019. [Online]. Available: http://tutorials.jenkov.com/javafx/tabpane.html

[47] ProgramCreek, "Java code examples for javafx.scene.control.togglegroup," 2020. [Online]. Available: https://www.programcreek.com/java-api-examples/?api=javafx.scene.control.ToggleGroup

[48] J. Jenkov, "Javafx datepicker," 2016. [Online]. Available: http://tutorials.jenkov.com/javafx/datepicker.html