

Thalia Project Report

Team Charlie

**Martti Aukia, Albert Boehm, Arthur-Louis Heath,
George Stoian, Daniel Joffe, Weronika Kakavou,
Marcell Veiner**



University of Aberdeen
Sunday 23rd February, 2020

Acknowledgement

TO DO: Thank stackoverflow and everyone else here.

Contents

| | | |
|----------|---------------------------------------|-----------|
| 1 | Introduction | 3 |
| 1.1 | Project Overview | 3 |
| 1.2 | Motivation/Rationale | 3 |
| 1.3 | Project management strategy | 3 |
| 2 | Background and Competitors | 4 |
| 2.1 | Portfolio Visualizer | 4 |
| 2.2 | Other Competing Software | 5 |
| 2.3 | Summary | 7 |
| 3 | Requirements | 8 |
| 3.1 | Functional Requirements | 8 |
| 3.2 | Non-functional Requirements | 10 |
| 4 | Design | 12 |
| 5 | Coding and Integration | 13 |
| 5.1 | Web Framework | 13 |
| 5.2 | Database Management System | 13 |
| 6 | Testing and Evaluation | 15 |
| 7 | Conclusions and further work | 16 |
| 8 | Appendices | 17 |

1 Introduction

1.1 Project Overview

This project is to create a portfolio backtesting software, which enables creating custom portfolios and measuring their performance with different backtesting functions. The user can pick from a variety of assets, some of which are Equities, Fixed Income, Currencies, Commodities and Cryptocurrencies. Then a quick visualization is performed for the allocated assets with risk and performance metrics.

1.2 Motivation/Rationale

Since retail investing is a growing market, our target audience consists of individual investors, who instead of seeking the full package that comes with financial advising, would like to take over the wheel and assess the viability of their investment strategies themselves. As retail investors are non-professionals and invest comparatively small amounts, with financial advice services being non affordable for those individual clients, we wanted to create a product that would not only be more affordable for small retail investors, but would also include a variety of international assets, which currently most backtesting software doesn't include.

1.3 Project management strategy

Team members are:

- Martti Aukia
- Arthur-Louis Heath
- Albert Boehm
- Marcell Veiner
- George Stoian
- Daniel Joffe
- Weronika Kakavou

The team held regular meetings, both with the project guide, Dr Nigel Beacham, and the Dr Ernesto Compatangelo. During the analysis stage meetings took place weekly and were aimed to discuss ideas and requirements. Whereas later, during the implementation stage the team held meeting at least once per week, some of which were aimed to discuss the design for the tool with the inclusion of some coding sessions.

2 Background and Competitors

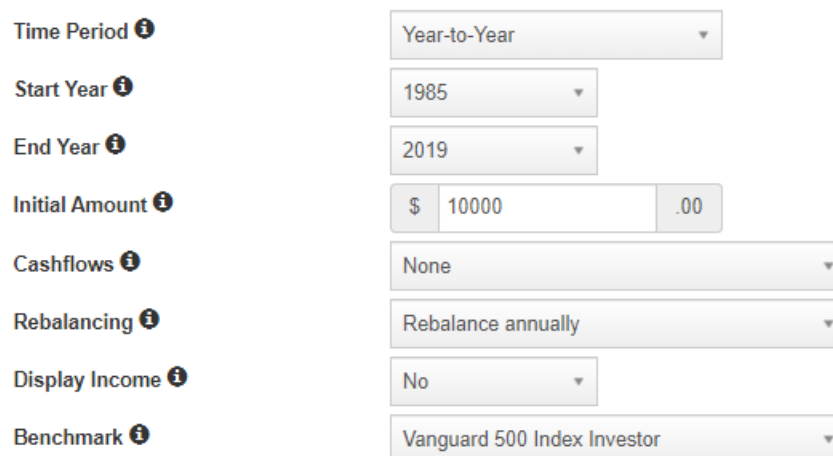
As a result of our initial competitor analysis, we may group the competing software into two categories. For a comprehensive list of available backtesters please see [?]. The first consists of various third-party trading software, such as Fidelity, MetaTrader and NinjaTrader that offer backtesting features as well. Although it may be beneficial to consider some trading features provided by the members of this group, these would only be part of a future development process, and not in the scope of this course.

The second category consists of pieces of software that do not offer trading as a service, and solely focus on backtesting. Thus, for now, we shall only consider the feature set provided by the second category. In the following paragraphs we will consider some of the design decisions made by competing software, together with the brief analysis and conclusion on each feature.

2.1 Portfolio Visualizer

Our main competitor of the second category is an online backtesting tool named Portfolio Visualizer[?]. During the inception phase of development we heavily relied on this website for writing the requirement analysis. Let us now briefly dissect what it has to offer.

Upon opening the website we are greeted with a brief description of the domain, together with the following input form:



The screenshot shows the input form for Portfolio Visualizer. It includes the following fields and their current values:

| Field | Value |
|------------------|-----------------------------|
| Time Period ⓘ | Year-to-Year |
| Start Year ⓘ | 1985 |
| End Year ⓘ | 2019 |
| Initial Amount ⓘ | \$ 10000 .00 |
| Cashflows ⓘ | None |
| Rebalancing ⓘ | Rebalance annually |
| Display Income ⓘ | No |
| Benchmark ⓘ | Vanguard 500 Index Investor |

Figure 1: Portfolio Visualizer - Input

As we can already see, the key elements of portfolio analysis are provided. Considering the functionality of our prototype as a baseline (that is testing an allocation of assets with a fixed initial investment on a fixed time period), we see that in addition users are able to do the following set of features:

- Set the endpoints of the investment period, up to months.
- Set the initial investment.
- Specify a regular cashflow and its frequency.
- Select a rebalancing strategy.
- Select a benchmark strategy for comparison.
- Compare multiple strategies at the same time.
- Adjust to inflation.
- Select from a set of lazy portfolios.

- Calculate additional metrics.
- Export the results to PDF, Excel, or save link.

At first it may seem as if Portfolio Visualizer meets all the requirements needed for a financial backtester, and indeed our main criticism is regarding the UI, responsiveness and user experience, and is a result of the initial user testing.

The UI design is simplistic and has a non-commercial, bare-bones look, and although this was appreciated while testing the system, it certainly does not improve the user experience. The input, mostly using dropdown menus is straightforward to use, except for the selection of Assets, which we will discuss briefly at the end of this section.

Moreover, users will want to fine-tune their investment strategies, by changing their allocations frequently. The only means to do this using Portfolio Visualizer is to scroll to the top, change the input and send it again. Our goal is to design a more responsive and dynamic system, to ease this procedure.

The website also has user accounts, these accounts however offer little to no extra features, which further decreases the overall user experience. Finally, as a last remark, which holds for most of the backtesting tools we have tried, the website is heavily US biased, and so the selection of asset classes is limited. Furthermore, we have no means of changing the currency, which would also be an important feature.

2.2 Other Competing Software

The rest of the alternatives are of a significantly lower quality. In the remaining parts of this section we will briefly consider a few design decisions made by these websites.

Tree-like Asset Selection

One of the tricky design decision in building the system, is the following:

What is the most intuitive way for selecting a portfolio item?

Where a portfolio item could be: an equity, ETF index, a commodity, bond or stock. Within these classes we have more options to choose from. Many backtesters opted for a search bar, which is a sensible approach, but poor in practice. Many portfolio items are named similarly, and for a new user it is a barrier, as they might not know what is available.

One of the better option is what ETFReplay [?] has implemented, which is a tree-like selection form, shown below.

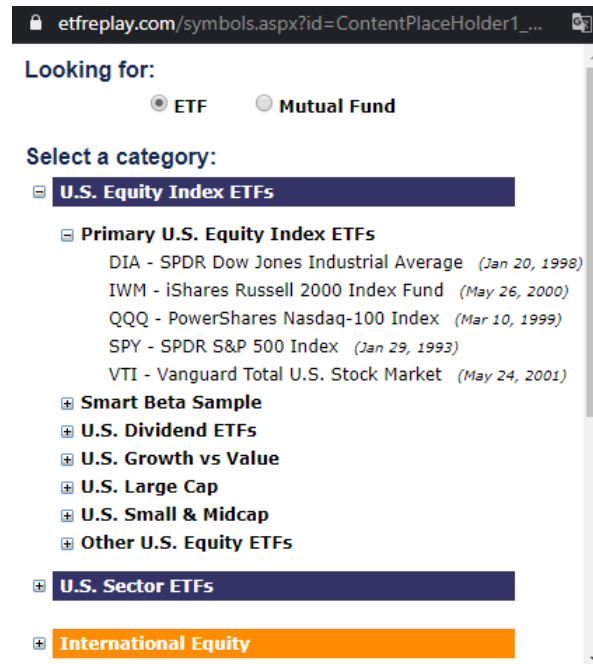


Figure 2: ETFReplay - Tree-like Structure

We feel this was the most intuitive to use, and will thus pursue a similar approach. Our only criticism was that it opens in a new window, which we would like to avoid.

Tiles

It may be worth briefly discussing an example of a backtester with a good layout design. We found the simplistic and tiled design of Backtest Curvo [?] a good choice as it gives the website an overall modern and fresh look, whereas most backtesters looked old and out of date. Our goal is to achieve a layout similar to this. For a further analysis on design decisions, please see section 4 of this report.

Simple Logic

In our whitepaper, we briefly discussed implementing a simple scripting language allowing users to simulate dynamic trading strategies. As mentioned, this would not be in the scope of the course, but for inspiration we can have a look at the approach of Stockbacktest [?].

Signals (Need at least one buy or short signal)
 Buy When:

| | | | | |
|-----------|------------------------------------|---------------------------|----------------|-------|
| Signal 1: | Upper Bollinger Band (days devs) ▼ | Crosses Below ▼ | Percentage % ▼ | AND ▼ |
| | 20 2 | none | 70 | |
| Signal 2: | Close Price ▼ | Is At Least % Above (%) ▼ | Percentage % ▼ | |
| | none | 15 | 10 | |

Figure 3: Stockbacktest - Simple Logic

Although not intuitive at all, given its presentation, it shows this feature is also possible. We believe that after nailing down how exactly the user would create the rules, this would be a straightforward task, as the calculations involved are not more complicated than in the static case. As mentioned above, this is something we would like to implement in the future.

2.3 Summary

As we initially tested the competing software during the inception phase of development, we were left with three key observations. These were the following:

- Transparency: Many of the testers we have found did not look transparent or trustworthy, which is something we should avoid.
- Learning Curve: After getting to know our target audience we concluded that most seem willing to learn how to use a complex system if it is worth it.
- Design of the UI: It was easy to tell which backtesters are still getting updated, by looking at their design. We should aim at looking fresh. In addition, some backtesters offered so many features that every corner of their UI was packed with information, this is something we should also avoid doing.

Having analysed the competing software, we are now better suited to determine the requirements of our system.

3 Requirements

We classify our requirements using the established FURPS+ model [1]. Below you will see our main functional and non-functional requirements. The items outlined below focus on some of the key requirements we have so far identified as features necessary to provide a compelling product for paying customers.

3.1 Functional Requirements

| Portfolio Configuration | |
|--|--|
| Allocate fixed amount/proportions of the portfolio to given assets | Choose how much each asset contributes to the portfolio's total value using either percentages or raw monetary amounts |
| Find assets quickly by category or name | When adding an asset the user can search a category for assets or search for a specific asset by its name |
| Share portfolio | Portfolios can be shared between people using a URL |
| Edit portfolio | Change asset allocation and their distributions in a portfolio |

| Portfolio analysis | |
|---|---|
| Compare portfolios | Use multiple portfolios in a single analysis to see differences in their performance |
| Use a selection of lazy portfolios | Select an existing common portfolio to compare against, such as common index funds (e.g. Vanguard 500 Index Investor or SPY) |
| Plot portfolio as a time-series | View portfolio performance as a line graph for a quick overview |
| Specify a time frame for the analysis | Select start and end dates for portfolio analysis |
| Choose rebalancing strategy | Optionally choose a strategy for buying and selling assets to meet your strategy e.g. buying and selling stocks each year to ensure the value of portfolio stays at 60% stocks and 40% bonds (i.e. maintain the initial allocation) |
| Change the distribution of assets in a portfolio using a slider | A slider for each asset to quickly increase or decrease its proportion of the total value |
| Edit portfolio analysis | Change parameters for portfolio's analysis after running it (e.g. date range or rebalancing strategy) |

| View results | |
|---|---|
| See key numerical figures | Show important numerical metrics for a portfolio's performance such as Initial Balance, Standard Deviation, Worst Year, Sharpe Ratio, and Sortino Ratio |
| See both real and nominal values | See portfolio's value as both adjusted and not adjusted for inflation |
| A breakdown of portfolio value at specific points of time | See what the value of the portfolio is at some point in time (e.g. January 3rd 1997) |
| Export result of analysis | Exports results to PDF for sharing and offline reading |

| User accounts | |
|-----------------------------------|---|
| Combine portfolios | Combine two portfolios' assets into one single portfolio |
| Save portfolio analysis for later | Save portfolio analysis parameters to the account so you can rerun it with a single click |
| Delete saved portfolio analysis | Remove a stored portfolio analysis from your account |
| Manage portfolio analyses | Edit saved portfolio analysis with different assets, distributions or other parameters |
| Sign-up, log in and log out | Basic authentication |

| Assets | |
|--|---|
| Choose assets from European market | Data for European assets were found to be lacking in competing products |
| Choose assets from Equities, Fixed Income, Currencies, Commodities, and Cryptocurrencies | Coverage of some of the largest asset classes |

3.2 Non-functional Requirements

1. Usability:

- The product must be easily usable for users who already have some financial investment experience.
- The basic backtesting interface needs to look familiar to people already experienced with it.
- The product must have detailed instructions on how to use its advertised functions.
- All major functions must be visible from the initial landing page.
- Must work in both desktop and mobile browsers.
- The results page should scale with mobile.

2. Reliability:

- The product must have a greater than 99% uptime.
- All our assets need to have up to date daily data where the asset is still publicly tradeable.
- All assets supported by the system must provide all publicly available historical data.

3. Performance:

- The website should load within 3 seconds on mobile [2].
- Large portfolios must be supported - up to 300 different assets.

4. Implementation:

- The system needs to work on a cloud hosting provider.

5. Interfacing:

- The Data Gathering Module must never use APIs stated to-be-deprecated within a month.
- The Data Gathering Module must not exceed its contractual usage limits.

6. Operations:

- An administrator on-call will be necessary for unexpected issues.

7. Packaging:

- The product needs to work inside a Linux container (e.g. Docker).
- All dependencies need to be installable with a single command.

8. Legal:

- All user testing must be done with ethical approval from the University.
- UI must display a clear legal disclaimer about the service not providing financial advice.
- All third-party code should allow for commercial use without requiring source disclosure (e.g. no GPL-3).
- User data handling should comply with GDPR.

4 Design

4.1 Data Segregation

The decision was made early on to horizontally partition the data store by Thalia into two parts. One consisting of data related to users and user accounts and the other of financial data related to asset classes, assets and their historical prices. The following is the list of reasons the team documented for this decision:

- One alternative revenue stream we identified early on was the sale of our financial data as a separate product. This process would be trivially easy if it was stored in a separate database.
- Although the security of both types of data is important to our business model, protecting user's private information is the highest priority. The financial data is accessed by the data harvester, a separate program gathering data from many sources on the web and introducing additional security risks. Data segregation helps limit the scope of a potential data breach.
- The two types of data serve two separate purposes. The modules responsible for managing each are also decoupled. Thus, separation helps to enforce the principle of least concern.
- A large corpus of guides and examples on how to manage user accounts is available online. Extending any of these to include financial data might be difficult, and risks leading to bad design.

The separation of dissimilar collections of data is a practice widely adopted in industry. Criteria for assessing when this approach is appropriate have also been documented. ?? Based on the decision to use SQLite as our DBMS and to maximize the portability and security of the financial data, we decided to implement this decision by using two separate databases.

4.2 The Finda Module

The Finda module was designed to implement the data layer, acting as an intermediary between the data harvester/business logic and the financial data. It allows users to manage a number of databases implementing a common schema and give them access to a suite of tools for reading, writing, and removing the data stored in each. In addition to this the Finda module implements the following features:

A system for managing user permissions to help reinforce separation of responsibilities among Thalia's other modules. Integrity checks to ensure the integrity of the data provided to the end user. A suite of administrative features to aid with managing the application back end

Finda's design was modeled after object relational mappers (ORMs), libraries offered by most popular web frameworks the use of which was prohibited by the project constraints. Although the implementation of what is essentially our own ORM proved to be costly in terms of developer time, it allowed us to create a more focused module tailored to our requirements. This helped to streamline the development of other modules.

5 Coding and Integration

5.1 Web Framework

One of the first decisions we had to make was which web framework to use. The two major options for Python are Django and Flask. Although we decided to use Django for our MVP in the first semester, we had to spend a relevant portion of the time available learning the framework, so we had to decide whether we were going to stick with it or learn Flask. In the end, we opted to go with Flask for the following reasons:

- Django has one architecture that all projects must share, and we have designed the architecture for our project ourselves. While neither architecture is wrong, the two are not compatible. Flask, on the other hand, is structure-agnostic, so we can lay out the code as we see fit.
- Flask comes with the bare minimum for web-development, which means that we don't need to manage the complexity of any feature we're not using. Django has a more complete feature-set from the beginning. This would be desirable in a large web application, but introduces significant overhead in our case, where the website has only a handful of pages.
- Django all but insists on using its ORM for all database interaction, while we plan to have a more manual approach.
- Our concerns were also confirmed by more experienced web-developers, suggesting simpler alternatives.

5.2 Database Management System

Another major technology decision was the choice of appropriate database management system (DBMS) for storing historical price data collected by the data harvester. Before committing to a specific technology we identified the following requirements a suitable DBMS should fulfil:

- **SCHEMA:** The structure of our data is relatively simple, consequently Thalia does not require support for sophisticated features and data types. A suitable DBMS should be able to accommodate the database schema designed last term, with the addition of simple integrity constraints and cascade operations.
- **SUPPORT:** Ideally the DBMS should be cross platform, as this would allow us to defer commitment to a specific deployment platform until we are ready to start the CD process.
- **LICENSE AND PRICING:** The DBMS should be free to use and have a non-restrictive license. **PERFORMANCE:** The DBMS should be able to handle a high volume of concurrent reads to fulfil user requests. The data will be updated daily, meaning efficient write operations are a lower priority.
- **USABILITY:** As our team lacks experience in this field, a suitable DBMS should be relatively simple to learn. Ideally team members should be able to learn the basics in a single weekly sprint.
- **SECURITY:** The DBMS should have a mature code base and be relatively secure, as access to financial data is a key component of our business model. Later it will likely also store data that is not available through public APIs, meaning potential data breaches could expose us to legal liability. ??
- **TYPE:** Since the project constraints specify we use SQL queries, only relational DBMS supporting a version of SQL are appropriate.

MySQL, PostgreSQL, SQLite and MariaDB were subject to in depth comparison based on fulfilment of the above requirements and industry adoption ?? ???. Our final decision was to use SQLite for the following reasons:

It is user-friendly and easy to deploy, allowing us to start continuous deployment faster. It has a small footprint and offers good performance. ?? Portable serverless design aids with development and testing. All team members have experience working with SQLite from previous term. This helps to reduce overhead of knowledge transfer.

The main drawbacks of using SQLite, namely scalability and performance are not a concern at this stage, as the current version of Thalia is meant to be a high quality industrial prototype, and as such will not contain the full range of financial data needed for marketability. Should SQLite prove to be inadequate in the future, we would be able to switch to a different DBMS with relatively little trouble, as the process of database migration is exceedingly well documented ?? ??. To preempt any difficulties that might arise, the decision was made to design the data layer to easily accommodate such a migration.

6 Testing and Evaluation

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

7 Conclusions and further work

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

8 Appendices

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.