Real Estate Portfolio Optimization Design Document

Blake Roberts	Project Lead / Backend
Colton Goode	Meeting Scribe / Backend
Kevin Johnson	Quality Control / Frontend
Leelabari Fulbel	Meeting Facilitator / Frontend
Nickolas Moeller	Report Manager / Backend
Client	Principal Financial Group
Faculty Advisor	Chinmay Hegde
Team Website	https://sdmay19-07.sd.ece.iastate.edu
Team Email	sdmay19-07@iastate.edu

Table of Contents

Figures

Tables

Symbols

Definitions

API	Application Program Interface
GUI	Global User Interface
HTTP	HyperText Transfer Protocol
MPT	Modern Portfolio Theory
MVP	Minimal Viable Product
REST	Representational State Transfer
SQL	Structured Query Language
SPA	Single Page Application

1.0 Introductory Materials

1.1 Acknowledgement

We would like to thank our advisor Chinmay Hegde. Also, Benjamin Harlander, Q Mabasa, Arthur Jones, Jonathan Ling, and Jonathan Frank of Principal Financial Group.

1.2 Problem statement

Principal lacks an in-house (non-3rd-party) tool capable of assessing the potential value and risk of many real estate property assets within a constrained portfolio. Principal's current strategy is to perform asset level analysis with all the market-level analysis to be assessed by a third party known as Costar. Costar compiles quarterly reports which are verbose. The reports are static and contain a lot of unnecessary information. So, the market-level analysis is time-consuming and offbase.

The most crucial aspect of the market-level analysis is that of portfolio optimization set forth by the ideas of Modern Portfolio Theory (MPT). A software capable of imploring such an optimization model as Markowitz or Black-Letterman would greatly benefit Principle. Moreover, a software solution would allow flexibility in constraining the optimization model to meet the precise desired needs of a portfolio investment team at Principal.

1.3 Operating environment

The frontend web application will be written in the Dash Python framework. It will interface via HTTP with a backend server written in Python. Data will be stored and queried via an SQL database. When users wish to use our application, they will run a Python script in their command prompt. Then, they will go to their preferred web browser and type in the port that the application is Flash server is running on.

1.4 Intended Users and Intended Uses

Primarily, the intended users of the application would be portfolio managers and portfolio analysts. They would load their portfolio data in and run our Markowitz portfolio optimization algorithm to see what their next investment may look like.

1.5 Assumptions and Limitations

1.5.1 Assumptions

1. The project users will have an understanding of portfolio optimization and the concepts of MPT
2. User's input data follows a standard format of real estate properties.

1.5.2 Limitations

1. A closed set of portfolio analysis will be available
2. Buying and selling real estate takes a long time due to the surveys that have to take place.

1.6 Expected End Product and Other Deliverables

The end product will be an internal software application that assists real estate portfolio managers in optimizing their investment portfolios. The users will be able to upload their current real estate portfolio, the expected returns, as well as various data constraints and assumptions about the market. The software application will use this data to determine a recommended optimized portfolio for our client to base their future investment decisions. Data visualization will be presented to the user via multiple graphs, charts, and other user-configurable visualizations.

2.0 Specifications and Analysis

2.1 Requirements

2.1.1 Functional Requirements

1. **Read current portfolio holdings.** The user will be able to upload portfolio holdings from CSV files. This would require standardizing the input format. Ideal: Read returns and covariance from a database.
2. **Read expected returns and covariance matrix for all assets.** The user will be able to upload returns and covariance matrix from CSV files. This would require standardizing the input format. Ideal: Read returns and covariance from a database.
3. **Update expected returns.** The user should have the ability to update the expected returns for specific asset groups by location, property type, or both.
4. **Define optimization constraints.** The user can add or edit constraints prior to optimization. See section on Optimization Constraints for desired functionality.
5. **Generate optimal portfolios.** Based on the user's defined constraints, the application will launch a backend process to determine the optimal portfolio holdings.
6. **Generate and visualize efficient frontiers.** The application will generate an efficient frontier from a range of risk & return values. It will show individual markets, property types, and the current portfolio compared to the frontier.
7. **Visualize current holdings.** The application will show current portfolio holdings by geography, property, etc. Show top N holdings. Summarize expected returns and risk.
8. **Visualize optimal portfolio results.** The application will show optimal portfolio holdings by geography, property, etc. It must show top N holdings and summarize expected returns and risk.
9. **Visualize differences between optimal and current holdings.** The application will summarize differences between the optimal and current portfolios and display in maps, charts, plots, etc.
10. **Recommend actions based on current holdings.** Recommendations for buy and sell decisions given the current portfolio holdings will be shown. The application will attempt to justify decisions based on increasing expected returns, reducing risk, or both.
11. **Share results.** Results will be exported to a report or share via email.

2.1.2 Non-Functional Requirements

1. The system will use only open source libraries and frameworks.
2. The system's reliance on specific types of infrastructure must be minimal.
3. Optimization takes no longer than 5 seconds to calculate.
4. A team of portfolio managers or analysts can use the application at one time.

2.2 Proposed Design

The project requires an application consisting of a user interface allowing the analysis and construction of models based on real estate market data. Reasonable proposed designs and approaches would be software solutions. As per the requirements of the project, both a native application and a web application are feasible. From a systems perspective, there are three key designs that have been considered.

The first proposed approach, suggested by the client, is a native application written entirely in R. The system would utilize the Shiny framework for the user interface. This application would be the most simplistic in design. It would, also, leverage the Principal data science team's current understanding of the language and GUI framework. However, this system would require the end user to download and run the application locally. Additionally, any future update to the application would need to be manually updated by the end user.

A second proposed application, deemed the most complicated, but with greatest scalability and maintainability, is a frontend/backend web application consisting of a client-side JavaScript SPA (created via Dash) and a Python server utilizing Flask to supply the client an HTTP REST API. Though this solution would be ideal for a group of software and computer engineers, the resultant system would require familiarity in many software technologies completely foreign to the Principal data science team.

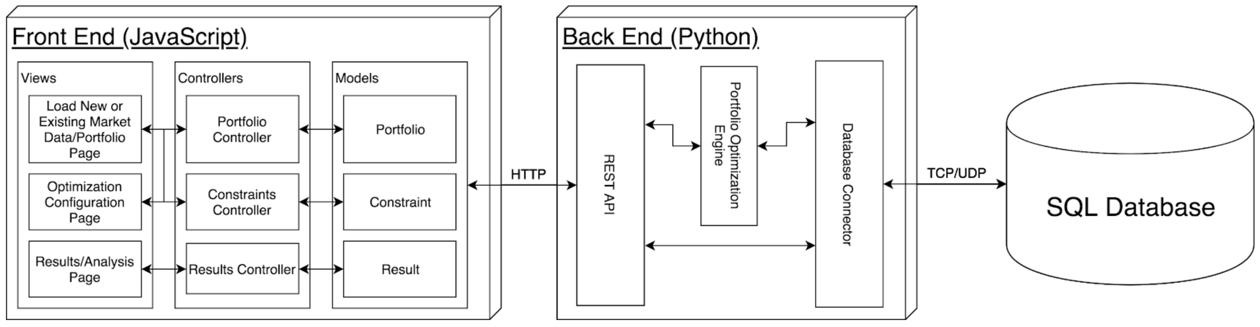

Figure 1: Application Block Diagram Utilizing a Server/Client Paradigm

A final approach would be to develop a Python Flask server integrated with the Python Dash framework by Plotly. The resultant web application would be hosted on Principal servers. This application would be reachable by any computer within Principals network via an internet browser. This design would only require knowledge in Python which makes it an ideal candidate for this project.

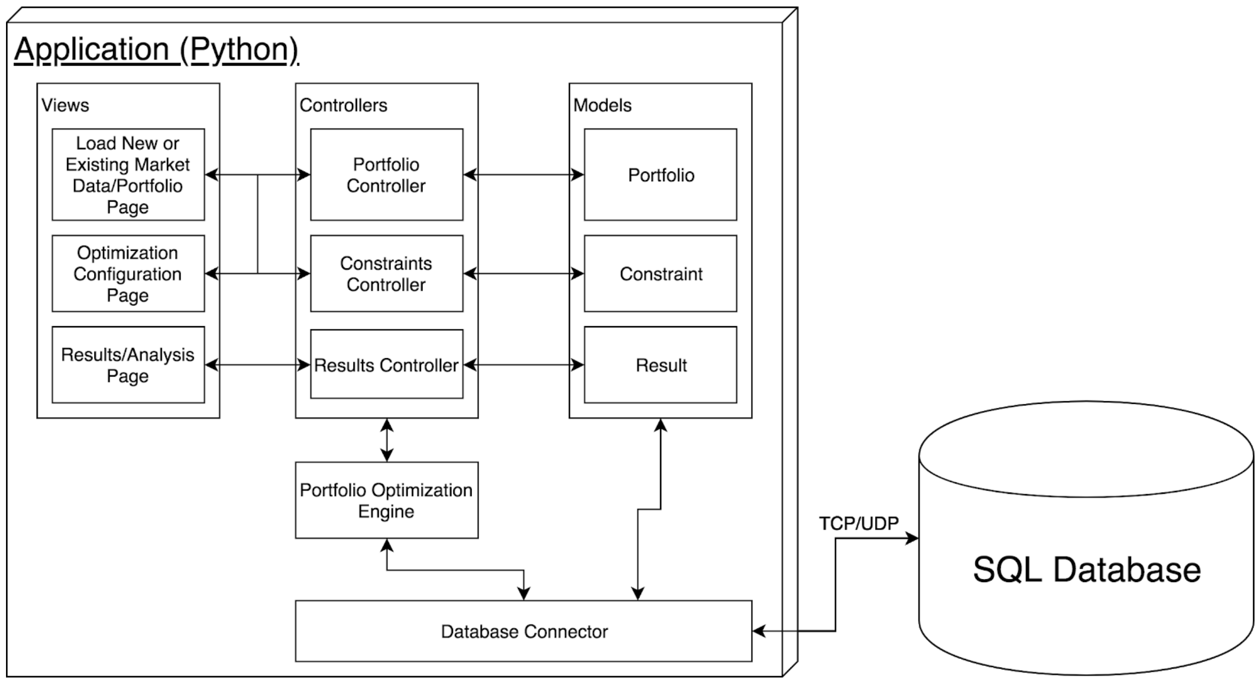

Figure 2: Application Block Diagram Utilizing Dash Framework

2.3 Design Analysis

A rudimentary Markowitz portfolio optimization model has been developed utilizing a test data set supplied by the client. The model was a first pass implementation with a statistical bias to be corrected in future iterations. The model utilized total return averages as reasonable expected returns.

The resulting portfolio weights from the first pass Markowitz model were reasonable. There are some faulty aspects of the implementation including user input. The algorithm makes static assumptions that will need to be extracted and readily adjusted through a user interface.

Taking a step back, the Markowitz model itself fails to realize many factors within MPT. A possible improvement on the application is to implore multiple optimization models such as the Black-Litterman model. The Black-Litterman model, developed by Fischer Black and Robert Litterman out of Goldman Sachs, is an improvement on the older Markowitz model. The Black-Litterman model is considerably more complex than the Markowitz model. For this reason, it will be important to begin development with the Markowitz model, but keep in mind the extensibility of the application to include additional models such as the Black-Litterman model.

2.3.1 Design Analysis Notes:

We have broken our group into frontend and backend teams. The design analysis of our backend team was on how the data analysis algorithms would be implemented. They made a decision to use Python due to conversations with the Principal data science team. While R is a faster programming language, it lacks the API capacity that Python does to interact with the frontend. The backend team researched how to calculate a naive approach to expected returns, a geometric mean of historical values. They are starting to implement a way for users to enter their own expected returns. For the frontend, the team sent out questions to the Principal real estate and data science team to get an idea on what the interface should look like. Principal uses PowerBI to visualize all of their data, as it allows predefined ways to present the data and built-in interactivity.

The Design Analysis was a successful endeavor for us. We were able to gather more information about what our clients would need and what they expect. We have a working optimization algorithm. We were able to start designing the mockups for the frontend based on the survey results and have built a prototype.

3.0 Testing and Implementation

3.1 Interface Specifications

The Python backend server will service a frontend built on top of Dash. Dash compiles to JavaScript but is written in Python. The interaction of the backend with the frontend will be done through a stateless HTTP API. The API serves as a contract in which both systems must adhere. To ensure functionality, two sets of integration tests for both systems have been developed.

The backend server integration tests utilize the unit-test and requests python packages to hit a running server instance with mock data and assert accurate and successful responses.

The front-end integration tests utilize the mock-server npm package to mock server responses and assert appropriate application behavior. The testing framework Jest will be used as well. The frontend will communicate with the backend for portfolio results.

3.2 Hardware and Software

This software application is designed to be cross-platform, as it is tested on Windows, Mac, and Linux operating systems. Therefore, it should not be hardware dependent, and only require minimal environment configuration beyond installation. It also is designed to mainly run off of built-in Python packages or open source libraries, meaning that installation and future modification by the client is easier in the future.

The software application will be hosted on the client's servers. These servers must meet certain minimal specifications to successfully run the application and serve the frontend application to browsers. To ensure the system is in working condition, InSpec by Chef, an open source testing framework for infrastructure, will ensure that the server is accurately configured to host our application. These tests can be run at deploy time to assert any faults at the infrastructure level.

The backend server will be designed using Flask to handle various requests from the frontend (the user). To test this, we have created tests to send predetermined requests to the server that mimics what the frontend would send as the user interacts with it. Once the request is analyzed, the server calls other backend code to perform analysis on the portfolio. As for the front end, which will ultimately be designed with Dash, will similarly take in pre-made endpoints to simulate data being passed to it for testing purposes.

3.3 Functional Testing

Backend unit tests are developed utilizing the unit-test Python package. The tests will consist of two major sections: endpoint tests, and optimization tests. The endpoint tests will assert that the various backend resource endpoints function correctly, i.e. return the proper information. The optimization tests will test the underlying optimization code including the Markowitz portfolio optimization model code and its components.

The frontend UI components, written utilizing the Dash Python framework, will be tested utilizing the unit-testing package as well. These tests will assert that required components are rendered on the correct pages via the correct endpoints or server routes.

Manual functional tests will also take place in the form of acceptance testing carried out by end users within Principal. These will be carried out after a functional Minimal Viable Product (MVP) has been developed. At this point, the development cycle will mimic a two-week agile sprint cycle that will allow iteration as end users give qualitative feedback from their acceptance tests.

3.4 Non-Functional Testing

3.4.1 Security

Our project will be run locally on the computers of Portfolio Managers. At some point after our time working with Principal, the REST API that we created may be placed on the servers of Principal, however, our primary goal is not for that. Since our project is only local, we have fewer security concerns. The brunt of the work for security will be dealt with by Principal as far as who gets access to portfolio files (in CSVs) which would include confidential client information.

3.4.2 Reliability

Because our application will be locally hosted on the computer of the PM, reliability does not need to be tested. If Principal hosts our REST API then they would have to check the reliability of their servers but that would not be our responsibility.

3.4.3 Availability

We will be doing health checks on the availability of our backend HTTP API.

3.4.4 Efficiency

We will run a series of test on our optimization scheme based on different sizes of portfolio CSVs and time the optimization for each size. This will allow us to determine the overall efficiency of our algorithm.

3.4.5 Scalability

Simultaneous unit tests on the backend will ensure performance doesn’t slow.

3.4.6 Usability

After we have an MVP, we are coordinating with Principal to have real users testing our project. This ensures our frontend is simple, easy-to-use, and reads the data from our backend correctly.

3.4.7 Interoperability

The application should work on Linux, Windows, and Mac operating systems. The application will also be able to be used along with the Dash desktop app through the Rest API that we have created.

3.5 Modeling and Simulation

Not applicable to our project.

3.6 Implementation Issues and Challenges

3.6.1 Implementation Issues

Translating the Markowitz model to Python code. We started meeting a second time every week to go over technical details; that helped us get the optimization done faster.

3.6.2 Challenges

Our first challenge was understanding the scope of the project. None of us have a background in real estate investing or MPT. The initial learning curve was very steep.

Another challenge was choosing whether to implement the Markowitz or Black-Litterman portfolio optimization models. After discussing with our client, Markowitz seemed like an easier, more feasible, introduction to the problem space. Black-Litterman was rendered a stretch goal for the project as a “nice to have” feature. The Black-Litterman model is an extension to the Markowitz model.

There has been challenge in finding the optimal tool to perform data visualization. First, we planned on using Dash to create everything but later learned about Power BI from Principal. Dash has been decided for the use of creating frontend using Python. Plotly has a security issue because when using it we would have to send data to Plotly servers. The portfolios that our application will use are confidential client information. This data should not be shared with an untrusted third party source. However, there does exist an offline version where that isn’t necessary. The data visualizations for the offline service do not seem to be as good. Power BI Embedded however does not allow for easy changes in the graphs. We would be limited to 48 API calls per day which is suboptimal and does not meet one of our non-functional requirements. We will instead be using Plotly offline which will create interactive visualizations for us.

4.0 Closing Material

4.1 Conclusion

Our goal is to create a software application that will allow real estate portfolio managers at Principal Financial to use data science to optimize portfolios while also allowing for the specification of particular interest areas for investment.

So far, our team has researched and completed multiple crucial steps in our project. We have a working Markowitz algorithm in place and are currently working towards allowing users to provide custom constraints into the optimization. Mockup designs for our frontend are also currently being translated to functional data visualizations for the user.

The current prototype of this application performs basic portfolio optimization for the user. Currently, both the frontend and backend components are functional for basic portfolio optimizations with a specific set of constraints.

In our proposed design we hope to add a set of key features. Users will be allowed to provide custom constraints into the optimization and improving the data visualization on the frontend. Users will be able to view a table of expected returns for each property in the portfolio. For each property, a user would be able to edit the expected return for that property in our proposed portfolio. We also plan on allowing our users to add or delete specific constraints. Then we would allow users to view the optimized portfolio based on the aforementioned options and the original portfolio via a set of graphs.

4.2 References

[1] T. Idzorek, A STEP-BY-STEP GUIDE TO THE BLACK-LITTERMAN MODEL. Ibbotson Associates, 2018.