Real Estate Portfolio Optimization Design Document

Blake Roberts	Project Lead / Backend
Colton Goode	Meeting Scribe / Backend
Kevin Johnson	Test Engineer / 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

Table of Contents **1**
- Figures 2
- Tables 2
- Symbols 2
- Definitions 2

Introductory Materials **3**
- Acknowledgement 3
- Problem statement 3
- Operating environment 3
- Intended Users and Intended Uses 3
- Assumptions and Limitations 3
  - Assumptions 3
  - Limitations 3
- Expected End Product and Other Deliverables 4

Specifications and Analysis **5**
- Requirements 5
  - Functional Requirements 5
  - Non-Functional Requirements 5
- Proposed Design 5
- Design Analysis 5

Testing and Implementation **7**
- Interface Specifications 7
- Hardware and Software 7
- Functional Testing 7
- Non-Functional Testing 7

Closing Material **8**
- Conclusion 8
- References 8

Figures

Tables

Symbols

Definitions

API	Application Program Interface
MPT	Modern Portfolio Theory
REST	Representational State Transfer
SPA	Single Page Application

Introductory Materials

Acknowledgement

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

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.

Operating environment

Frontend web application is written in JavaScript which interfaces via HTTP and/or TCP with a backend application written in Python. Data will be stored and quired via an SQL database. As of now, the application will be written in

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.

Assumptions and Limitations

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 that will be defined later in the project.

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.

Expected End Product and Other Deliverables

Our team is expected to deliver a software capable of performing rudimentary portfolio optimization. Furthermore, the software will display and compile summaries and reports regarding what results the software provides.

Specifications and Analysis

Requirements

Functional Requirements

1. The system will provide a user interface to allow customization of the portfolio optimization model.
2. The system will store repeated custom procedures.

Non-Functional Requirements

1. The system will use only open source libraries and frameworks.

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.

A second approach as devised as the simplest system by the team is a static web application written in Python utilizing the Flask framework for application routes.

A final approach deemed the most complicated, but with greatest scalability and maintainability is a frontend/backend web application consisting of a client-side JavaScript SPA and a Python server utilizing Flask to supply the client an HTTP REST API.

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 extendability of the application to include additional models such as the Black-Litterman model.

Design Analysis Notes:

So far we can broken our group in parts to focus on the dfferent aspects of the project. We have abackend and frontend focused groups. The design analysis of the backend focused group was on the decision between 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 then looked in to how to based expected returns from the historical actual return of averages. For the Frontend, the team sent out questions to the Principal data science team to get an idea on what the

The Design Analysis was a sucessful endeavor for us.We were able to gather more information about what our clients would need. We were able to start designing the mockups for the frontend based off the survey results.

We were able to approve our ability to plug in better expected returns, along with other options

Testing and Implementation

Interface Specifications

The Python backend server will service a frontend JavaScript client, or browser. This interaction 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 unittest and requests python packages to hit a running server instance with mock data and assert accurate and successful responses.

The frontend 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 mostly be using CoffeeScript as an MVC framework. It will communicate with the backend for portfolio results.

Hardware and Software

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 ran at deploy time to assert any faults at the infrastructure level.

Functional Testing

Backend unit tests are developed utilizing the unittest Python package. Coverage.py is used in parallel to measure code coverage.

Frontend unit tests utilize Jest, a JavaScript testing framework.

Along with frontend unit tests, TestCafe, another JavaScript testing framework, is used for UI testing.

The plan will also do live tests with Principal for acceptance testing.

Non-Functional Testing

Service availability tests like health checks. Could do performance tests in the form of unit tests on the Backend.

Closing Material

Conclusion

So far, our team has managed to complete a couple crucial steps. We have a working Black-Letterman algorithm in place. We have started to make mockup designs for our frontend.

Our goal is to create an app that will allow clients of Principal to use data science to optimize portfolios while also allowing for the specification of particular interest areas for investment.

The best plan of action for us is to create a native application that has a frontend and backend component. The backend will written in a Python framework. That is the best option for allowing portofolio optimization while still incorporating the ability to communicate with the frontend framework. The Principal team is most familiar with it so they would be able to continue developing on it in the future. We will be using CoffeeScript for our frontend. This choice is more ambigious and just has to deal with our personal preferences.

References

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