



SPRING 2019 PORTFOLIO OPTIMIZER

TOWSON UNIVERSITY AIT 715 CASE STUDY

JASON TUCKER
ADVISED BY JAL IRANI



TABLE OF CONTENTS

1. Abstract	2
2. Technologies Used	3
3. Application Architecture	5
3.1 Overview	5
3.2 Microservice Descriptions	6
3.3 Microservices Communication Flow	6
3.4 Portfolio Optimization Workflow	7
3.5 Asynchronous Parallel Processing	7
4. System Features	8
4.1 Use Cases	8
4.2 User Access Control	9
4.3 Portfolio Creation	13
4.4 Pending Results	15
4.5 View Results	16
4.6 Optimization Results Management	22
5. Optimization Methodology	25
6. Benchmark Comparison Methodology	33
6.1 Beta	33
6.2 Capital Asset Pricing Model (CAPM)	33
6.3 Jensen's Alpha	33
7. Project Management Process	35
7.1 Issues	35
7.2 Sprints	35
7.3 Kanban Board	35
7.4 Git Process and Continuous Integration	36
8. Challenges Encountered and Lessons Learned	38
9. The Future for Portfolio Optimizer	40
11. References	42

1. ABSTRACT

Portfolio optimization is the process of mathematically determining the optimal assets weights of a portfolio of financial assets to meet a specific objective.

Portfolio optimization was pioneered by Harry Markowitz with his “modern portfolio theory” mathematical framework. This framework was introduced in 1952 and is still highly influential today. His framework incorporated covariance of asset returns into calculations of portfolio returns and the standard deviation of the returns, which allows for calculations of risk adjusted returns [2]. This provides a mathematical approach to diversification; assets that have weak or inverse correlation with each other are more likely to volatility the risk of the portfolio than correlated assets.

The problem with performing portfolio optimization is that it usually done using tedious excel spread sheets or expensive proprietary software. Portfolio Optimizer solves these problems by offering an easy to use free application where users can perform portfolio optimizations.

The specific optimization objectives that are utilized in the Portfolio Optimizer project are:

- **Maximum Sharpe Ratio:** Sharpe ratio is a measurement of a risk-adjusted performance of a portfolio [1]. The creator of the Sharpe Ratio, William Sharpe, was a student of Harry Markowitz [2]. See the “Optimization Methodology” section for more information on the Sharpe Ratio.
- **Maximum Returns:** Achieve maximum returns of a portfolio while keeping the risk (measured by the standard deviation of the returns) constrained to the risk of a safe asset. The lowest standard deviation of returns among all assets in the portfolios will be selected as the constraint and the optimizer will produce the portfolio with the highest returns while staying at the low standard deviation.
- **Minimum Standard Deviation:** Achieve the minimum standard deviation of returns of a portfolio while keeping the returns constrained by the returns of a highly profitable asset. The highest average rate of return of all assets in the portfolio will be selected as the constraint and the optimizer will produce the portfolio with the lowest standard deviation while remaining at the high rate of return.

2. TECHNOLOGIES USED

Python 3.7: Python is an interpreted, high-level, general-purpose programming language.

<https://www.python.org/>

Select Python Packages:

- Celery: An asynchronous task queue/job queue based on distributed message passing. Portfolio Optimizer uses this to allow for parallel, scalable, and asynchronous execution of optimization jobs. <http://www.celeryproject.org/>
- NumPy: A library for creating high performance arrays and matrices, with a large collection of high-level mathematical functions supported. Portfolio Optimizer uses this to perform multiple linear algebra calculations to generate matrices for the optimizers. <https://www.numpy.org/>
- SciPy > Optimize > Minimize: Minimization of scalar function of one or more variables. Portfolio Optimizer's financial asset allocation optimizers are built using this tool. <https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html>
- Flask: A lightweight web framework, ideal for REST APIs. <http://flask.pocoo.org/>
- Gunicorn: A Python Web Server Gateway Interface HTTP server using a pre-fork worker model. <https://gunicorn.org/>
- yahooFinancials: A python package for retrieving data from Yahoo Finance. <https://github.com/JECSand/yahoofinancials>
- Swagger: A tool that helps developers design, build, document, and consume REST Web services. Swagger documentation is generated by a Flask extension, Flask-RESTPlus. <https://flask-restplus.readthedocs.io/en/stable/>
- BCrypt: A utility for encrypting passwords, and validating passwords against their stored encrypted form. <https://pypi.org/project/bcrypt/>

Angular 7: Angular is a TypeScript-based platform for building mobile and desktop web applications. Angular is a frontend web framework ideal for making single page applications.

<https://angular.io/>

Select Node Package Manager (NPM) Packages:

- TypeScript: A strict syntactical superset of JavaScript that adds optional static typing to the language. <https://www.typescriptlang.org/>
- Sass: A cascading style sheets (CSS) preprocessor, that allows for advanced creation of CSS files. <https://sass-lang.com/documentation/syntax>
- Highcharts: Interactive JavaScript charts for web pages. Portfolio Optimizer uses the Highstocks extension of Highcharts to generate Stock Charts. <https://www.highcharts.com/>
- Angular Material: Material design components for Angular. <https://material.angular.io/>
- Bootstrap 4: Front-end CSS framework. Portfolio Optimizer leverages Sass to have only select aspects of Bootstrap 4 imported, and it is themed to aesthetically match the Angular Material Theme. <https://getbootstrap.com/>

JSON Web Tokens (JWT): A compact URL-safe means of representing claims to be transferred between two parties. Portfolio Optimizer uses JWT to authenticate requests made to protected routes on Flask. <https://jwt.io/>

RabbitMQ: Message-broker software that implements the Advanced Message Queuing Protocol. <https://www.rabbitmq.com/>

MongoDB: A NoSQL documented-oriented database that uses JSON-like documents. <https://www.mongodb.com/>

Docker: A tool that performs operating-system-level virtualization, used to run isolated software packages called containers. <https://www.docker.com/>

Nginx: Nginx is an asynchronous, event-driven web server which can also be used as a reverse proxy, load balancer, mail proxy and HTTP cache. <https://www.nginx.com/>

3. APPLICATION ARCHITECTURE

3.1 OVERVIEW

The architecture of Portfolio Optimizer contains eight Docker containers, with each running its own microservice. These containers communicate with each other using HTTP or through shared Docker Volumes. The containers share a Docker Network; a private network within the docker environment.

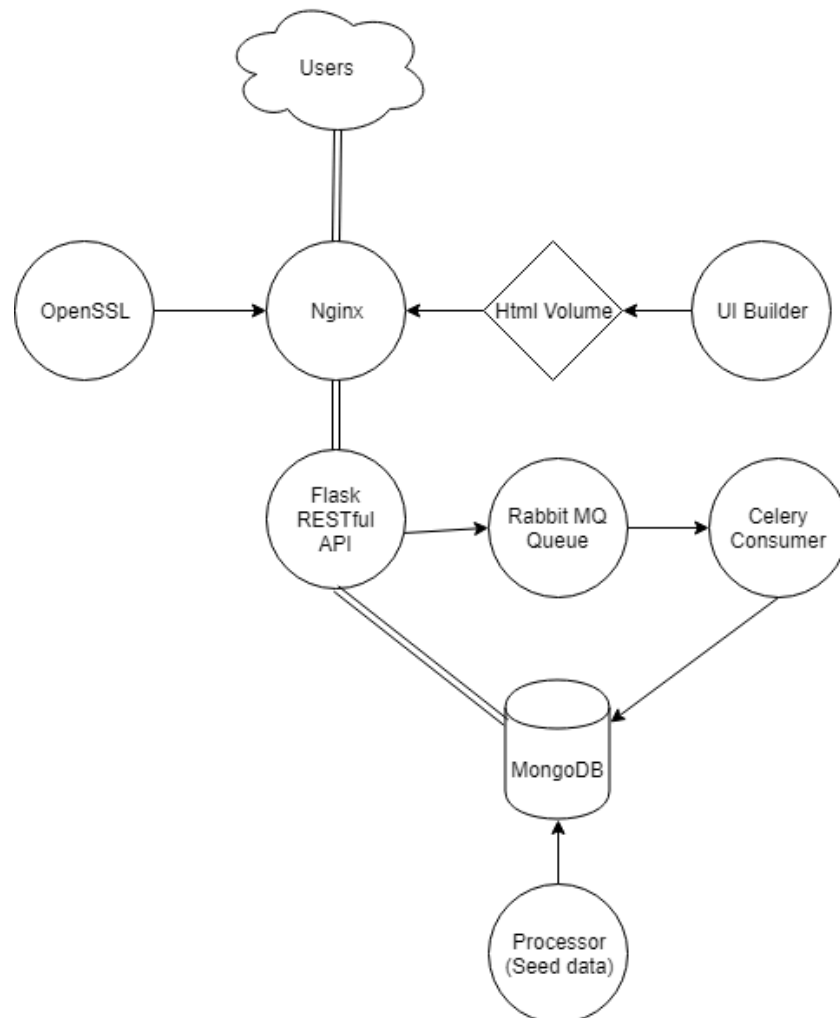


Figure 1: Application Architecture

3.2 MICROSERVICE DESCRIPTIONS

- Nginx (po-gateway): Nginx web server
 - Server static files for UI
 - Reverse proxy to Flask RESTful API
 - In a production deployment of the application, this container would have the only ports exposed outside of the Docker network
- UI Builder (po-build-ui): Compiles the Angular source code to be served by Nginx
- Openssl (po-openssl): Generates self-signed server certificates at startup for Nginx to serve HTTPS traffic
- Flask RESTful API (po-api): Flask RESTful API, served by Gunicorn
 - Uses shared backend image
- RabbitMQ (po-task-queue): RabbitMQ queue and management server
- Celery consumer (po-celery): Celery worker to process optimization tasks
 - Uses shared backend image
- MongoDB (po-database): MongoDB server
 - There is docker volume mounted (mongo_data_vol) to the container so the data does not disappear if the container is removed
- Processor (po-processor): Python script to upload data to MongoDB at startup
 - Uses shared backend image

3.3 MICROSERVICES COMMUNICATION FLOW

- UI builder shares files with Nginx through a Docker volume (html_vol)
- Openssl delivers certificates to Nginx through a shared host bind mount
- Nginx delivers data from an externally exposed port (80/443) to the Flask RESTful API's internally exposed domain and port using a reverse proxy
- The Flask RESTful API communicates with the Celery consumer by placing tasks in the RabbitMQ task queue
- The Celery consumer communicates with the Flask RESTful API by storing the state and results of tasks and in MongoDB
- The processor seeds MongoDB with data

3.4 PORTFOLIO OPTIMIZATION WORKFLOW

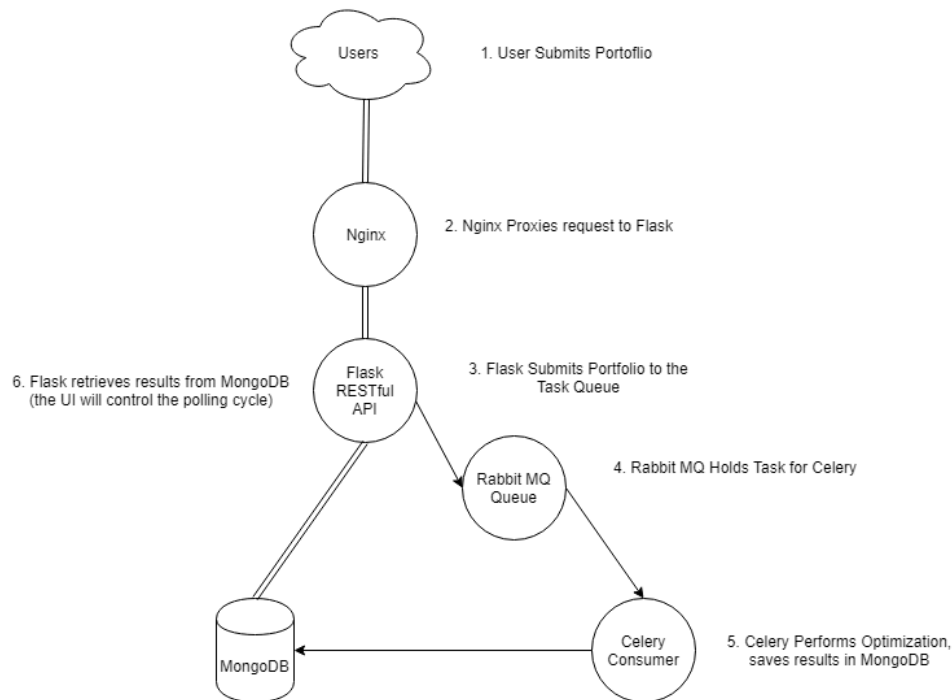


Figure 2: Portfolio Optimization Workflow

3.5 ASYNCHRONOUS PARALLEL PROCESSING

Portfolio Optimizer is designed to be highly scalable by using a task queue and worker architecture. It is designed such that the Flask RESTful API is never doing long or heavy processing. The heaviest task in the application is the portfolio optimization, which takes about 1.75 seconds to complete (significantly longer depending on the portfolio's parameters.) The Flask RESTful API will submit a task to the RabbitMQ task queue, then the Celery worker will consume the task and do the heavy processing.

Currently the application will run the Celery worker with a pool of 2 processors, meaning that two jobs can be processed concurrently. The results are made accessible to the Flask RESTful API by using MongoDB. This architecture is highly scalable because the pool size per worker could be increased, and the number of Docker containers running Celery workers could be increased, which would dramatically increase concurrent processing potential.

The downside to this approach is that both the application architecture and processing workflow becomes more complex. Due to the asynchronous nature of the workflow, it is not possible to submit an optimization task and view the results in a single HTTP request. The current solution is to submit the optimization task, poll for the task being completed, then request the results once they are ready.

4. SYSTEM FEATURES

4.1 USE CASES

4.1.1 Unapproved User

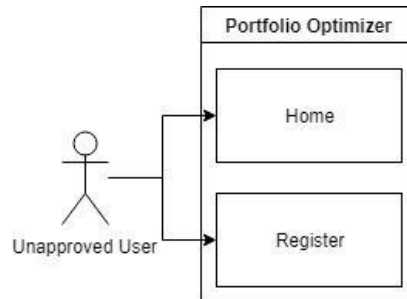


Figure 3: Unapproved User Use Case

4.1.2 Standard User and Administrators

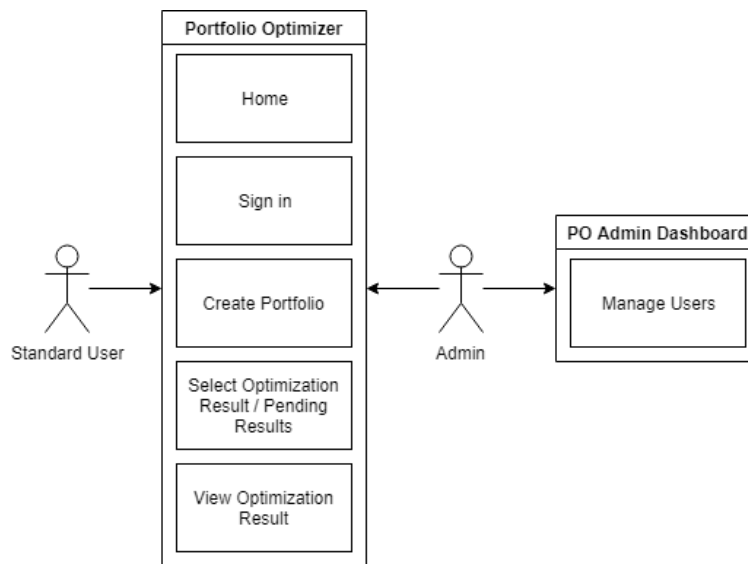


Figure 4: Standard User and Admin use cases

4.1.3 Angular Router Tree

The application is set up as a single-page application, thus the server responds with a single HTML file and Angular does all additional routing. The Angular router tree visualization was generated automatically by the Augury chrome extension.

Angular has nested/child routers, so the routing trees in Angular will be more complex than server side rendered sites. Note that “no-name-route” nodes represent auto-redirecting routes, or child routers without a root level route.

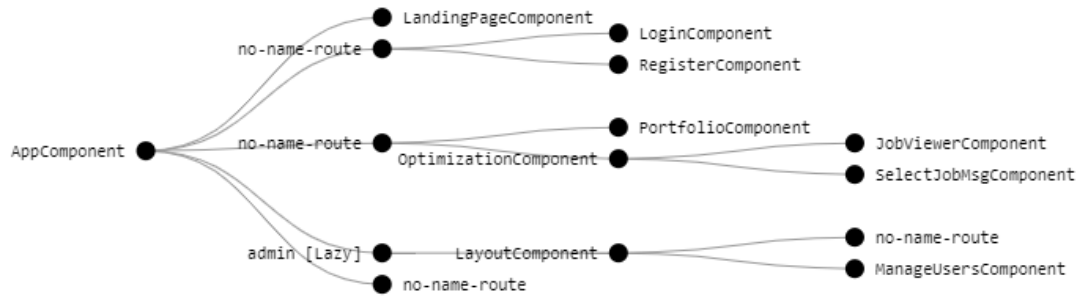


Figure 5: Angular router tree

4.2 USER ACCESS CONTROL

4.2.1 User Roles

There are two user roles in Portfolio Optimizer:

- Administrator: Have access to the “Admin” dashboard, where they can manage users.

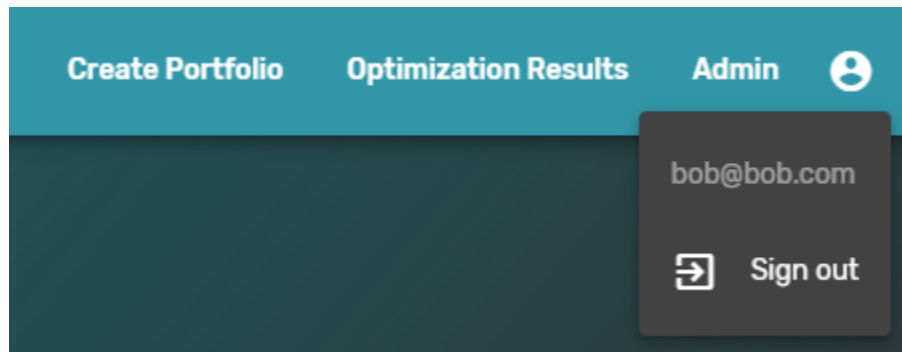


Figure 6: Navigation bar as Admin role

- Standard Users: Have access to the “Dashboard” section, which contains “Create Portfolio” and “Optimization Results” sections.

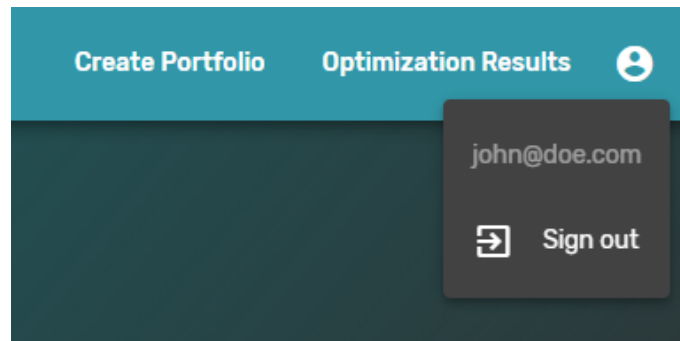
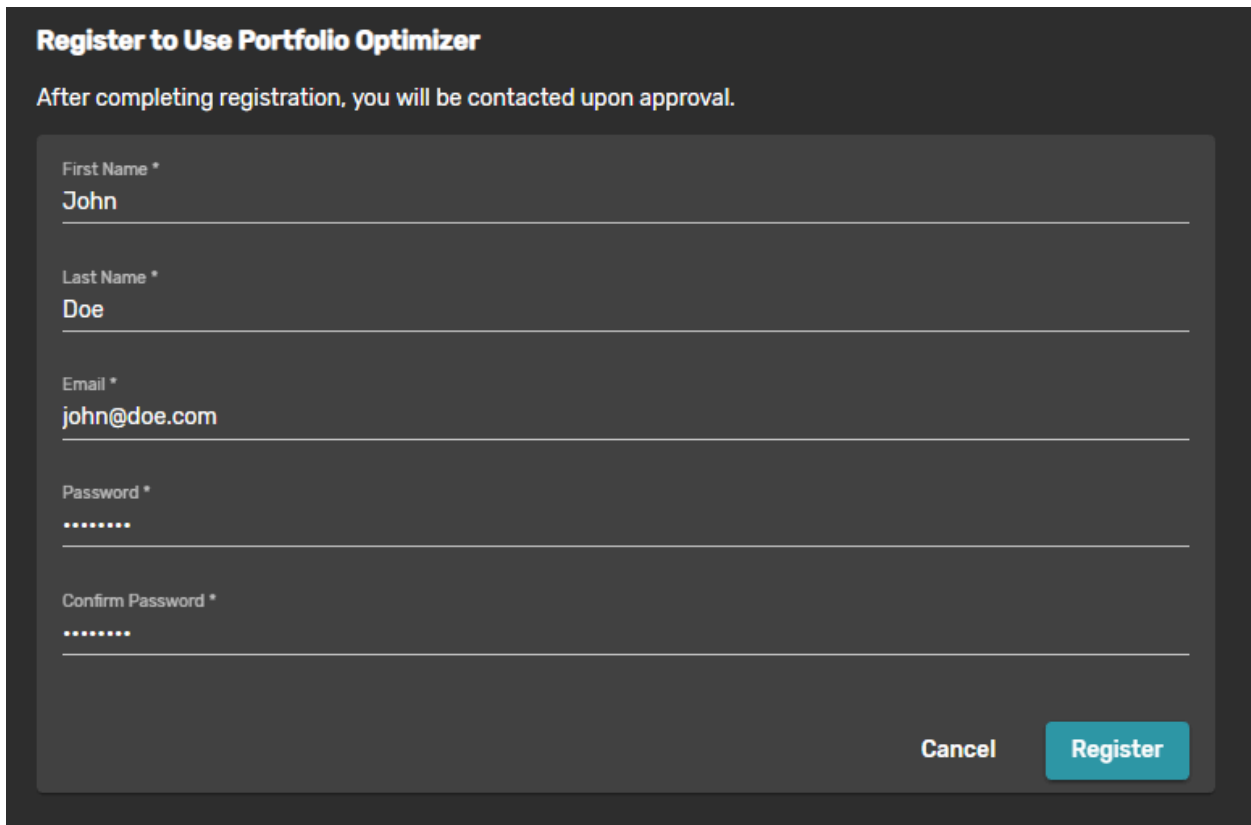


Figure 7: Navigation bar as Standard User role

4.2.2 Registration

New users to the application will go to the “Register” page and complete the registration form.



Register to Use Portfolio Optimizer

After completing registration, you will be contacted upon approval.

First Name *
John

Last Name *
Doe

Email *
john@doe.com

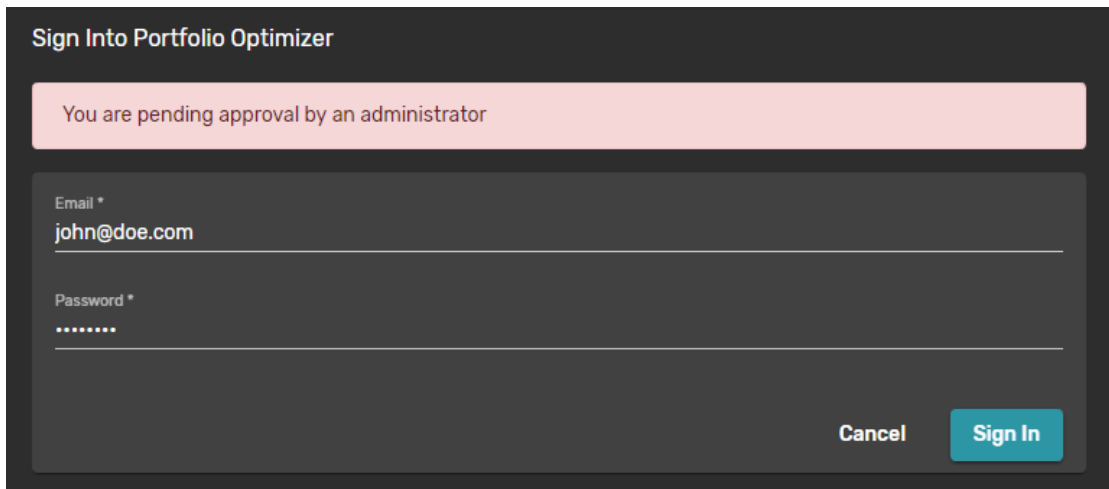
Password *
.....

Confirm Password *
.....

Cancel Register

Figure 8: Registration Form

Once the registered form is complete, the user will be entered into the system as an unapproved user. If the user attempts to sign in before being approved, he/she will see the following error message:



Sign Into Portfolio Optimizer

You are pending approval by an administrator

Email *
john@doe.com

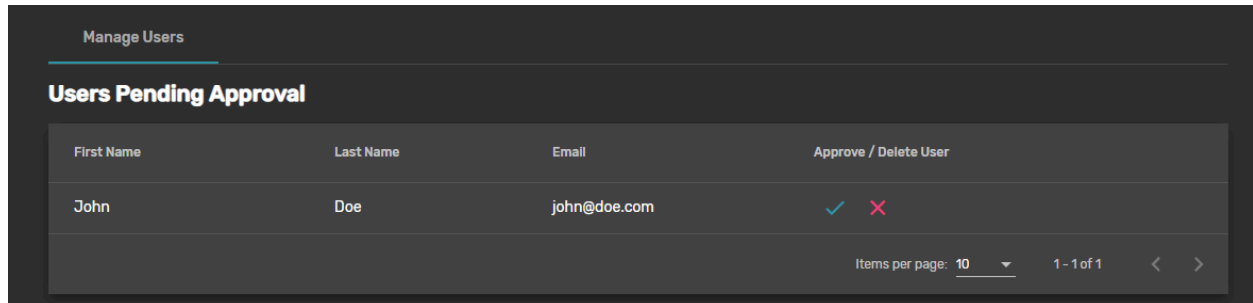
Password *
.....

Cancel Sign In

Figure 9: Sign in form pending approval error

4.2.3 User Approval

The administrator dashboard of Portfolio Optimizer has a “Manage Users” tab where users pending approval may be approved or denied access to the application:



The screenshot shows the 'Manage Users' tab with a sub-header 'Users Pending Approval'. Below it is a table with columns: First Name, Last Name, Email, and Approve / Delete User. There is one row for a user named John Doe with email john@doe.com. The 'Approve / Delete User' column contains a green checkmark and a red X. At the bottom right, there is a pagination control showing 'Items per page: 10' and '1 - 1 of 1'.



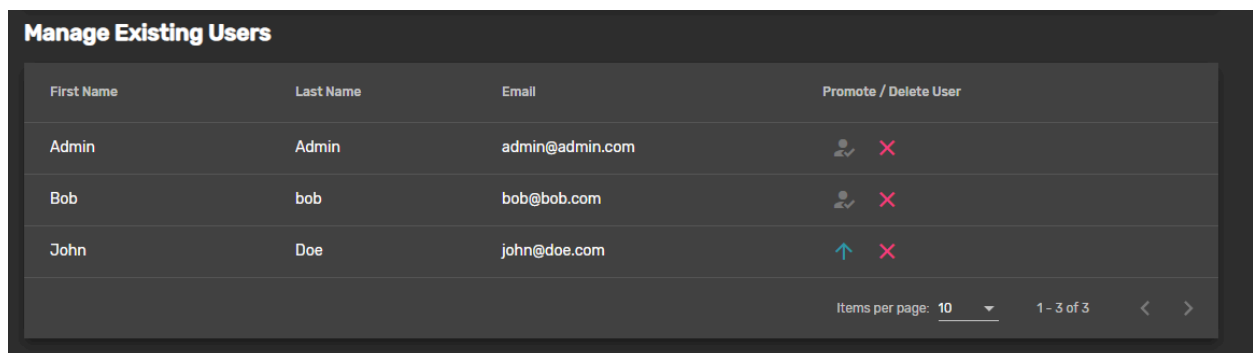
First Name	Last Name	Email	Approve / Delete User
John	Doe	john@doe.com	 

Figure 10: Users pending approval table

Once the user is approved (defaulted to standard user role), the account will show up in the “Manage Existing Users” table of this tab where the user may be deleted from the system, or promoted to an administrator.



The screenshot shows the 'Manage Existing Users' tab with a table containing three users: Admin, Bob, and John Doe. The columns are First Name, Last Name, Email, and Promote / Delete User. The 'Promote / Delete User' column contains icons for promoting (a person with a checkmark) and deleting (a red X). John Doe's row also includes a blue up arrow icon. At the bottom right, there is a pagination control showing 'Items per page: 10' and '1 - 3 of 3'.










First Name	Last Name	Email	Promote / Delete User
Admin	Admin	admin@admin.com	  
Bob	bob	bob@bob.com	  
John	Doe	john@doe.com	  

Figure 11: Manage existing users table

4.2.4 Authentication Mechanics

The Flask REST API has a “/user/login” route that verifies that the user exists, that the password is correct (using BCrypt to hash the password), and that the user is approved. If those conditions are met, the server generates an encrypted JSON Web Token and attaches it to the response. This token contains the user ID and role. This data in the token is encrypted using a password that only the server access to; the browser can read the expiry, but not the data.

Upon a successful login, the Angular web application will place the token in a globally accessible user service. Whenever an API call is made from Angular, an Angular feature called “HTTP Interceptors” is utilized to attach the token to a header that the Flask RESTful API is expecting the token to be in.

Protected Flask routes will decode the token and confirm it is valid and not expired. Admin-only routes will also check the Admin role stored in the token.

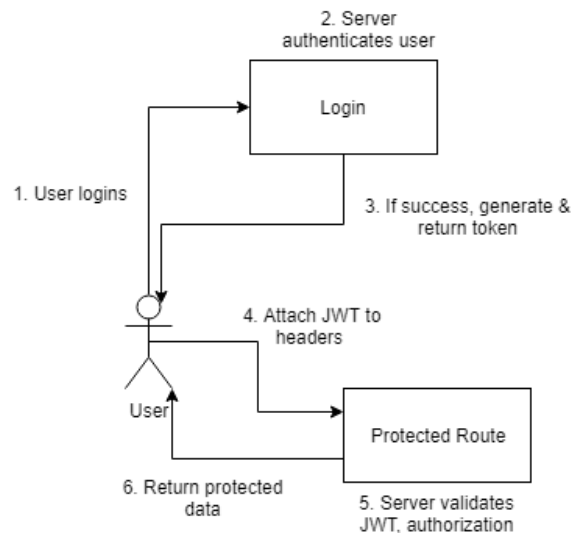


Figure 12: Process of an approved user accessing data from protected RESTful API routes

4.3 PORTFOLIO CREATION

4.3.1 Overview

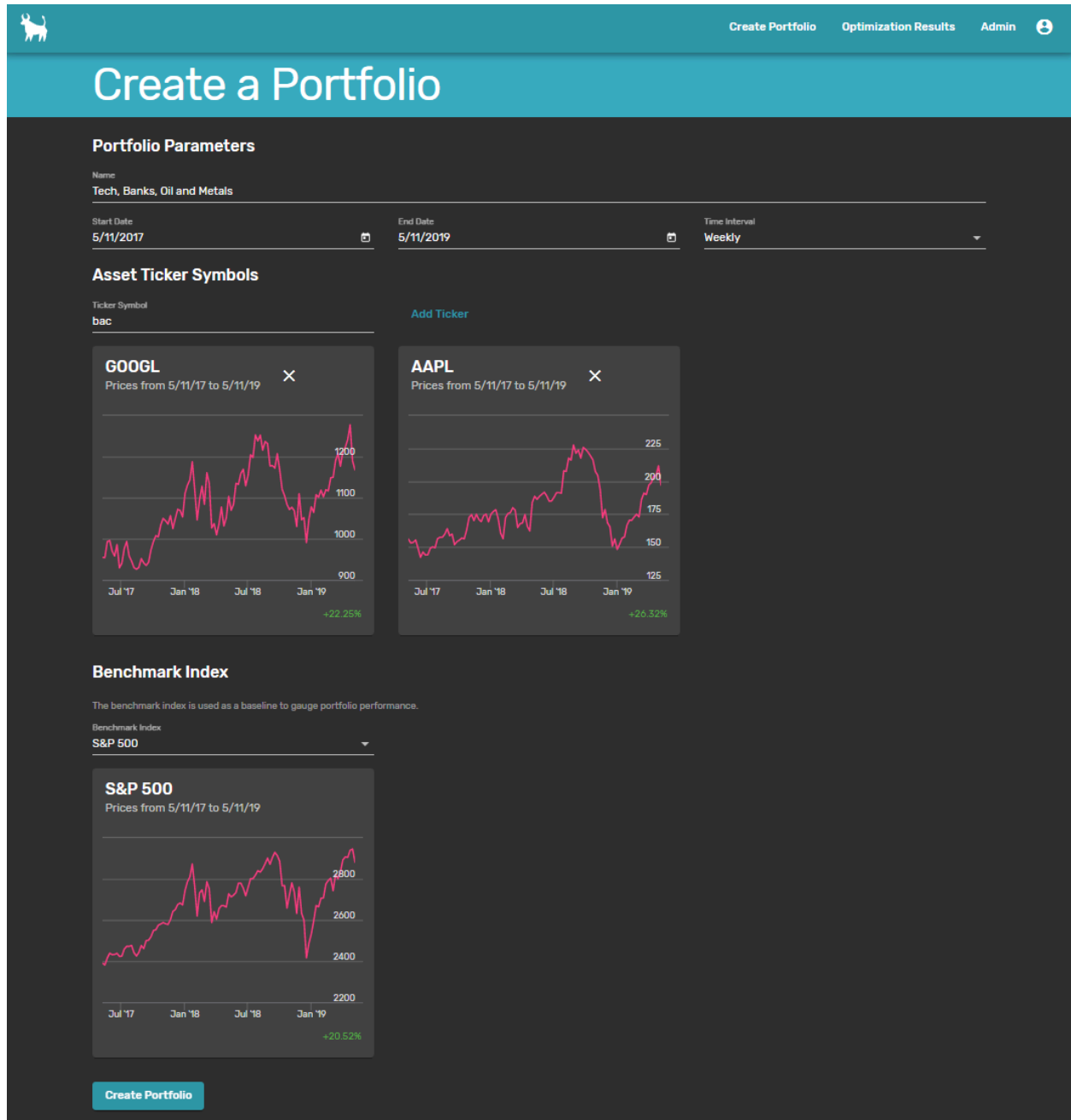
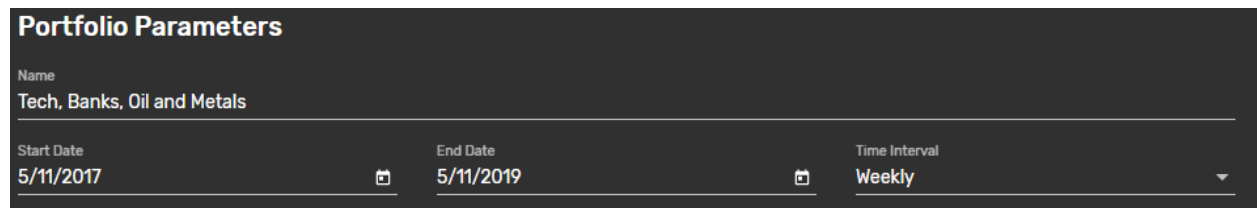


Figure 13: Create portfolio overview

4.3.2 Portfolio Parameters

The “Create Portfolio” section of the application is where users create setup the portfolios that will be optimized. The first section, “Portfolio Parameters” is where users name the portfolio

and set the date range and interval of prices to consider. The date range and interval are used by the “yahooFinancials” library on the backend to bring in data matching those parameters for each asset added.



The form is titled "Portfolio Parameters" and has a dark background. It contains three input fields: "Name" with the value "Tech, Banks, Oil and Metals", "Start Date" with the value "5/11/2017", and "End Date" with the value "5/11/2019". There are calendar icons next to the date fields. To the right of the date fields is a "Time Interval" dropdown menu set to "Weekly".

Figure 14: Portfolio parameters form

4.3.3 Adding Assets to the Portfolio

The “Asset Ticker Symbols” section is where the user will enter the ticker symbols of the assets that he/she wishes to add to the portfolio. A price chart of each asset added will be displayed, and it will be update if the date range changes. This is so the user can quickly see how the assets performed over the time specified by the date range. Additionally, users may remove any asset from this list if they change their mind about including it.

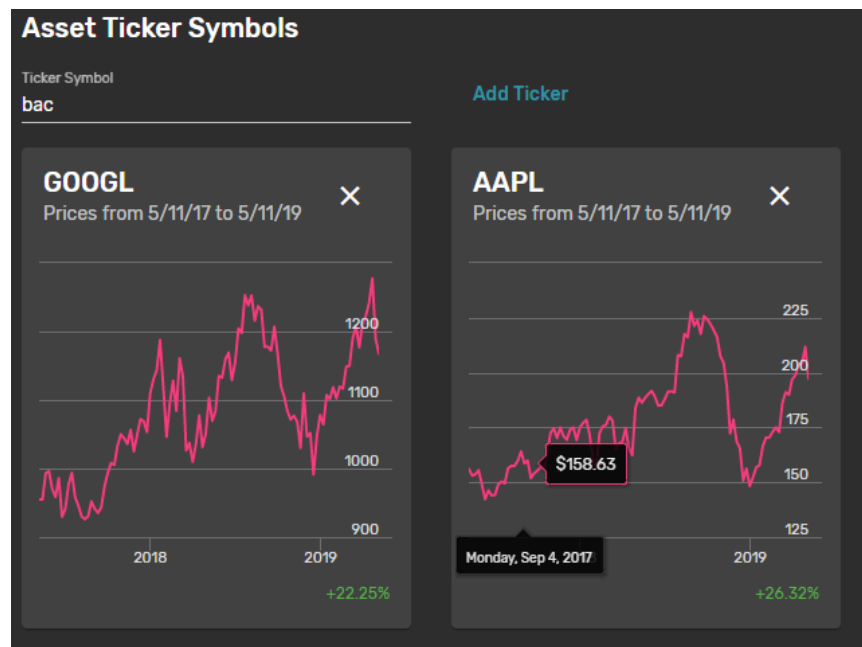


Figure 15: Asset ticker symbols form and visualizations

4.3.4 Benchmark Index

The “Benchmark Index” section is where the user selects a common stock index that will be used to generate comparisons in the optimized portfolio.

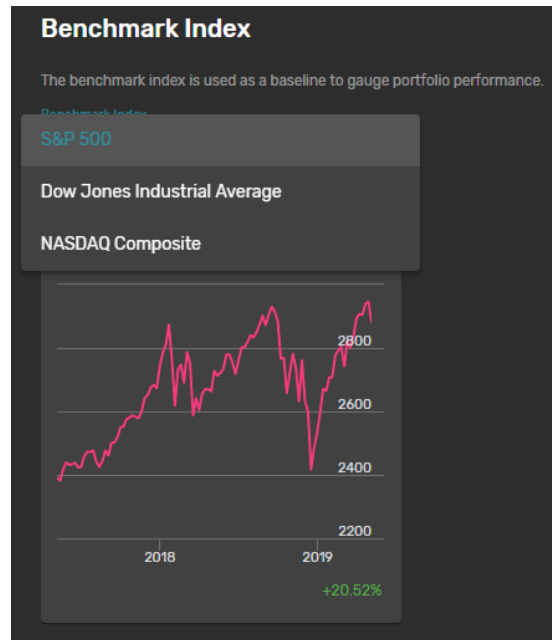


Figure 16: Benchmark index form and visualization

Please see the “Benchmark Comparison Methodology” section for more information about how benchmark indexes are used in the application.

4.4 PENDING RESULTS

Upon submitted a portfolio, the user is redirected to the “Select a Portfolio” section of the “Optimization Results” dashboard. The recently submitted portfolio will be placed into a “Pending Portfolios” list. The portfolio optimization is performed asynchronously on the backend; while the job is being processed, the status of the pending portfolios will be polled and updated, with the user having to refresh the page.

Pending Portfolios	
Name	Task State
Tech, Banks, Oil and Metals	<div></div>

Figure 17: Pending portfolio table with incomplete task

Pending Portfolios	
Name	Task State
Tech, Banks, Oil and Metals	Complete

Figure 18: Pending portfolio table with completed task

Once the portfolio optimization is complete, it will be added to the “My Portfolios” list and be given a “New” badge to draw attention to itself.

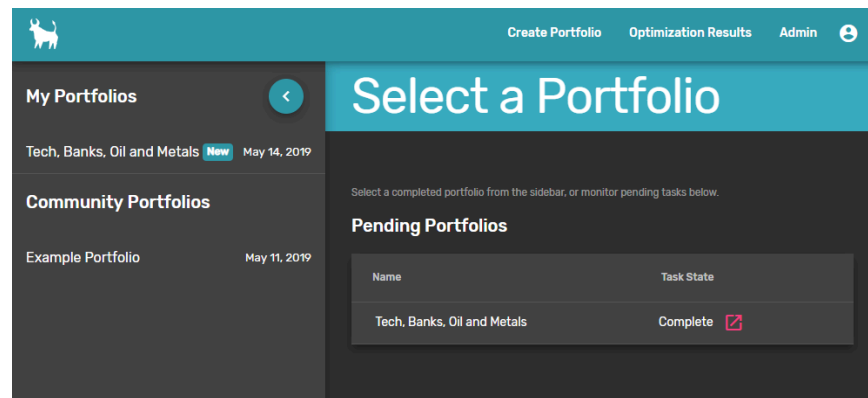


Figure 19: Completed task appearing in navigation menu with “new” badge, without page reload

4.5 VIEW RESULTS

4.5.1 Overview

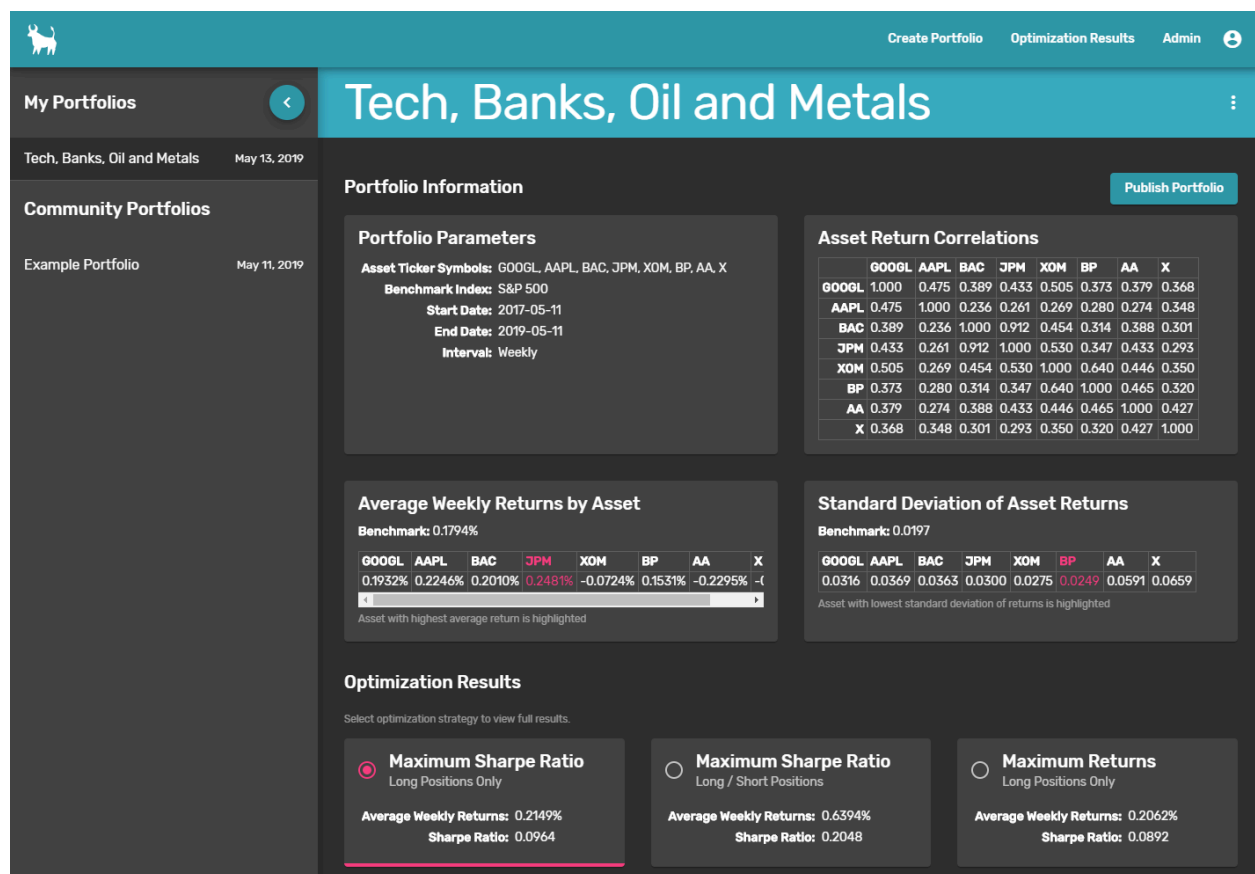


Figure 20: View results overview

4.5.2 Portfolio Information

The “Portfolio Information” section of the results page shows the parameters of the portfolio, and general statistics about the assets in the portfolio. One of these panels is the “Asset Return Correlations” matrix, which shows how correlated each asset is with every other asset.

Asset Return Correlations								
	GOOGL	AAPL	BAC	JPM	XOM	BP	AA	X
GOOGL	1.000	0.475	0.389	0.433	0.505	0.373	0.379	0.368
AAPL	0.475	1.000	0.236	0.261	0.269	0.280	0.274	0.348
BAC	0.389	0.236	1.000	0.912	0.454	0.314	0.388	0.301
JPM	0.433	0.261	0.912	1.000	0.530	0.347	0.433	0.293
XOM	0.505	0.269	0.454	0.530	1.000	0.640	0.446	0.350
BP	0.373	0.280	0.314	0.347	0.640	1.000	0.465	0.320
AA	0.379	0.274	0.388	0.433	0.446	0.465	1.000	0.427
X	0.368	0.348	0.301	0.293	0.350	0.320	0.427	1.000

Figure 21: Asset return correlations table

4.5.3 Optimization Results Selector

The “Optimization Results” section shows all of the optimization strategies that were ran against the portfolio, as well as a “Equal Weights” strategy (allocating an equal weight in every asset) that acts as a control group for comparing other strategies. This shows a limited number of statistics about each strategy to help indicate whether further investigation of the strategy is warranted.

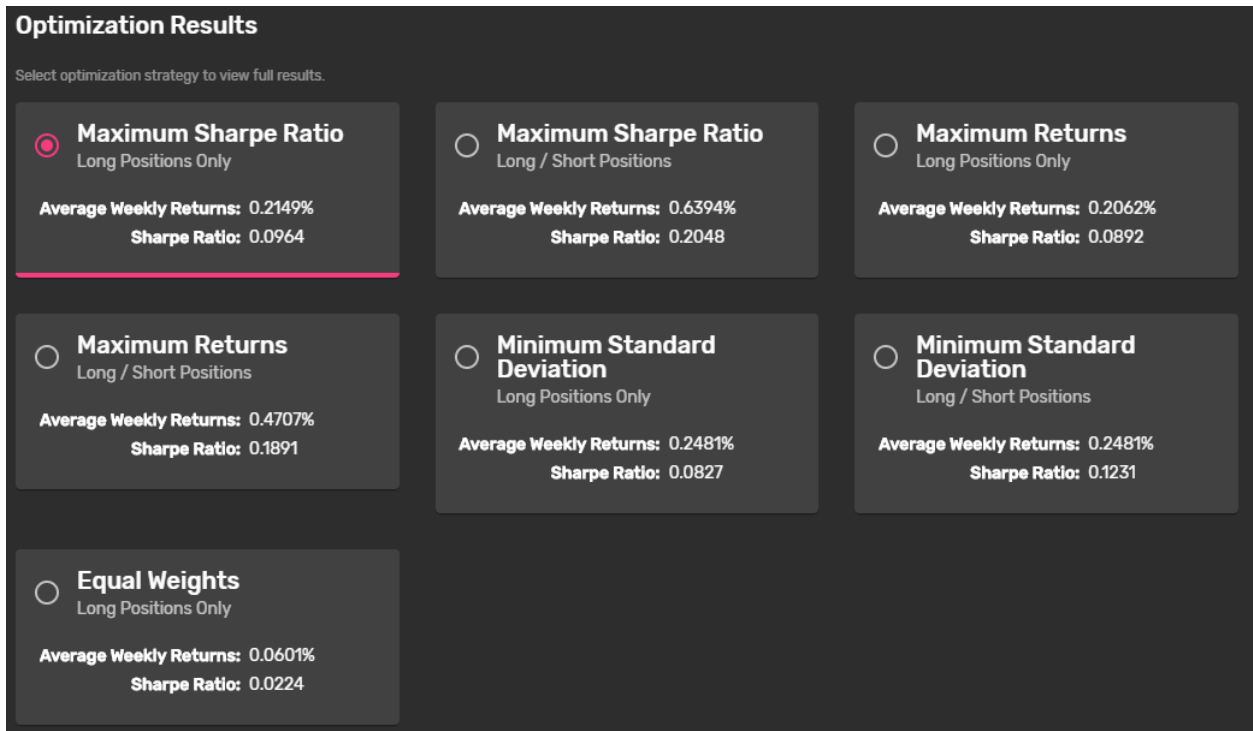


Figure 22: Optimization results navigation and summary information section

This section also acts as the navigation for the strategy details section, as clicking on a strategy will display the details of that strategy.

4.5.4 Optimization Strategy Details

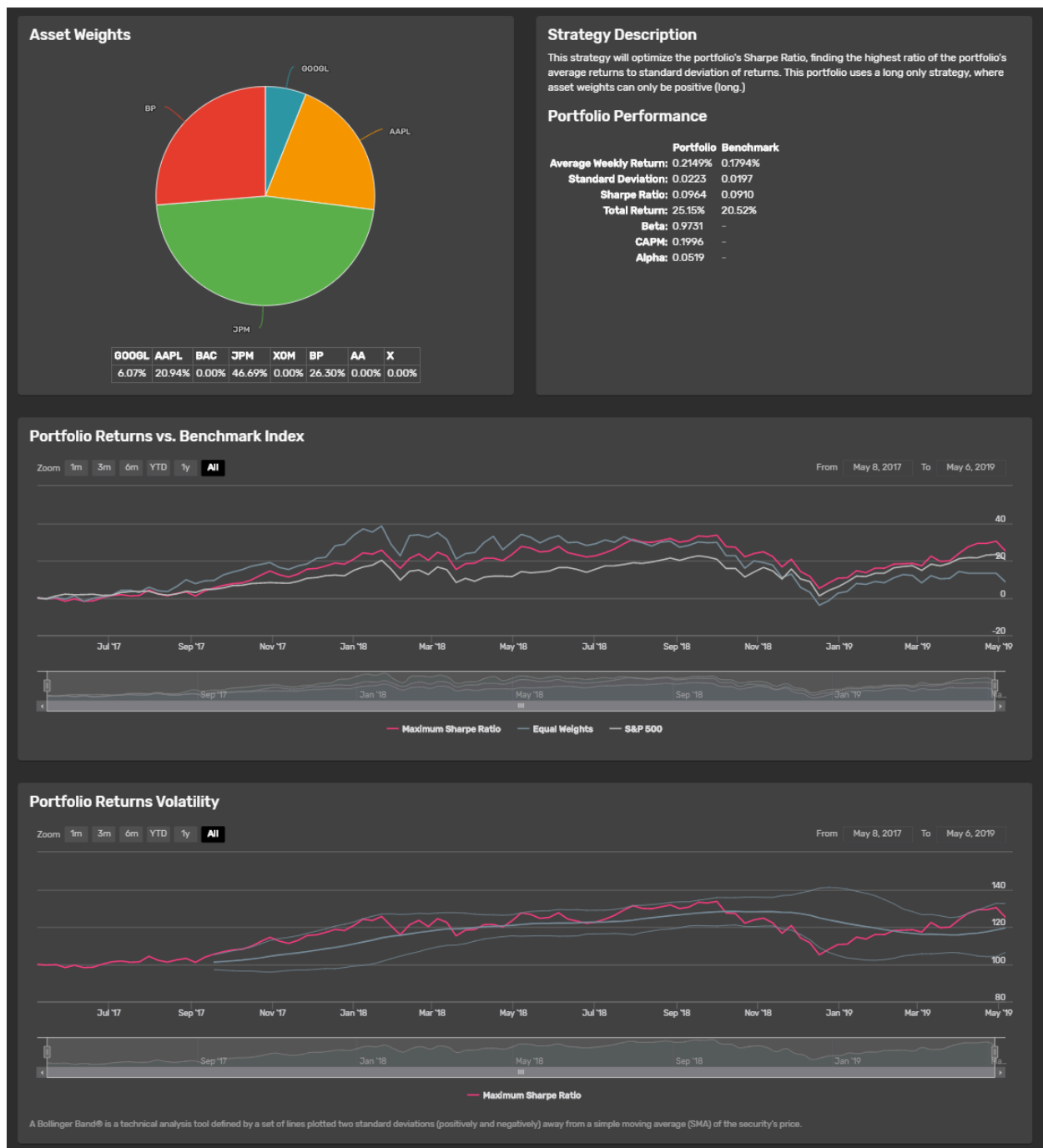


Figure 23: Selected optimization strategy overview

The “Asset Weights” panel shows the weights of the assets that the optimizer assigned. This is displayed in both a pie chart and a table.

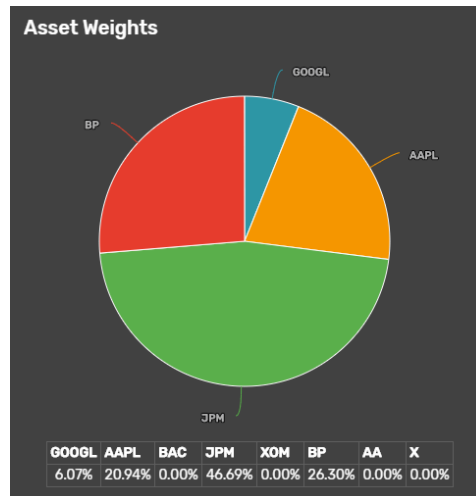


Figure 24: Selected optimization strategy asset weights with a long only strategy selected

If the strategy is a long / short strategy, asset weights can be either positive or negative. In this case, the pie chart indicates whether if the position is long or short with a “+” or “-”:

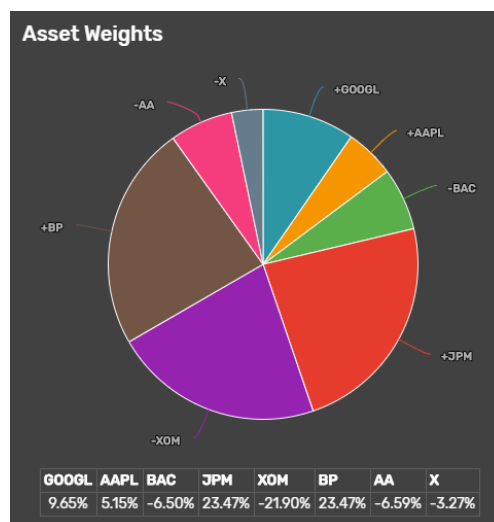


Figure 25: Selected optimization strategy asset weights with a long / short strategy selected

The information panel shows additional information about the strategy, beyond what is shown in the “Optimization Results (Selector)” section. Some of the metrics shown only in this section include standard deviation of asset returns, total return over date range, beta, capital asset pricing model (CAPM), and alpha. Additionally, information about the benchmark index is shown next to these statistics (where applicable) to allow for easy comparison against the benchmark performance.

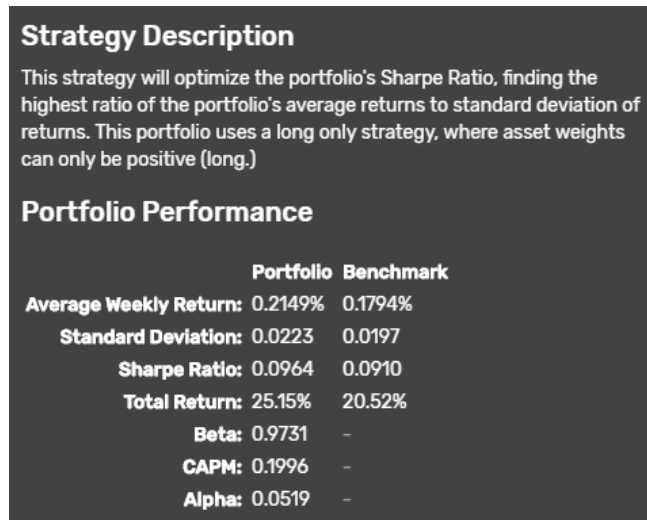


Figure 26: Selected optimization strategy description and performance statistics

The “Portfolio Returns vs Benchmark Indices” panel contains an interactive stock chart where the returns of the portfolio overtime are compared against the benchmark index and the equal weights strategy.

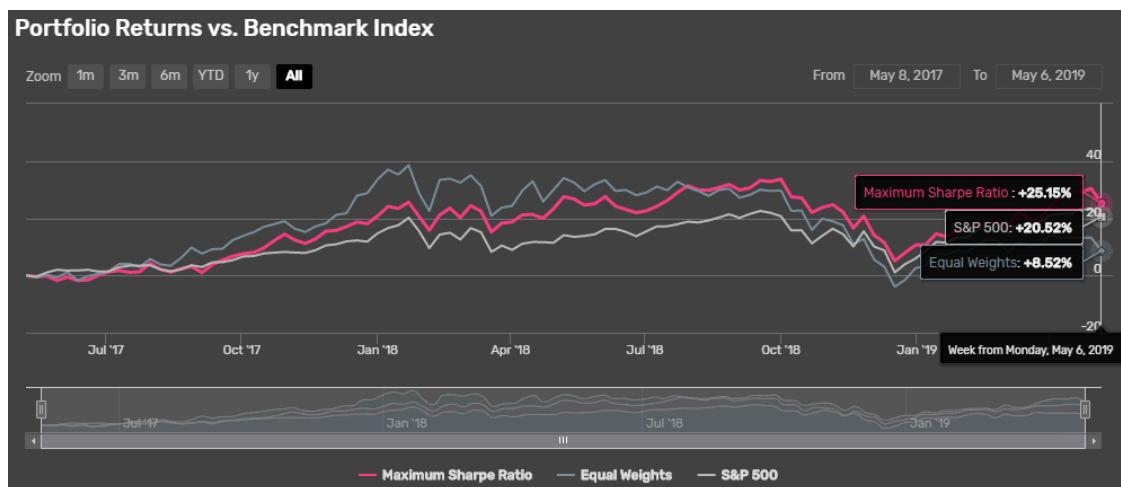


Figure 27: Selected optimization strategy performance during optimized date range, relative to benchmark indices

The “Portfolio Returns Volatility” panel contains an interactive stock chart where the asset returns are graphed along with the Bollinger bands technical indicator. Bollinger bands is a combination of a simple moving average line (the average price over X period), as well as two standard deviations above and below the simple moving average. This is a quick way for users to see whether if the portfolio had periods of high volatility, which would be readily apparent by widening Bollinger bands.

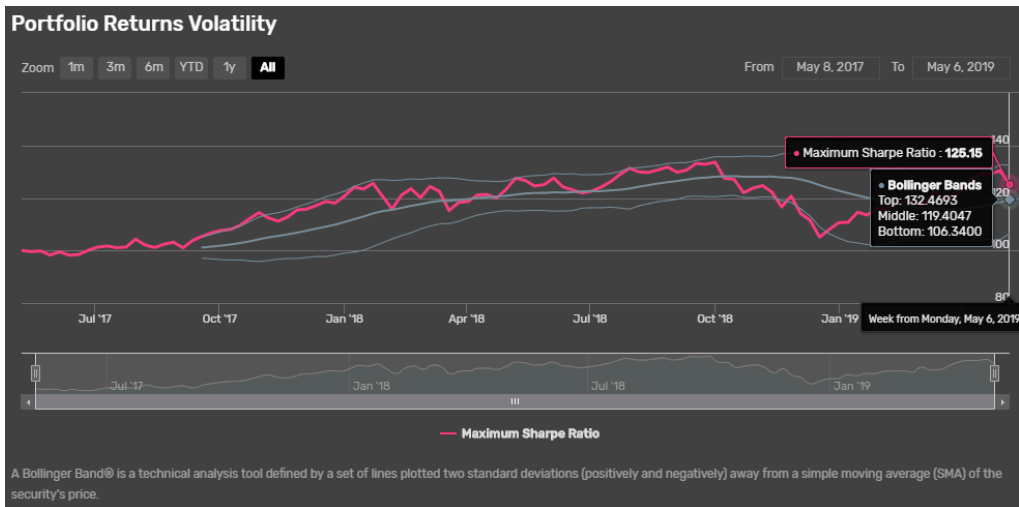


Figure 28: Selected optimization strategy performance during optimized date range alongside Bollinger Bands

4.6 OPTIMIZATION RESULTS MANAGEMENT

Each optimization result portfolio can be managed within the application. Each management tool can either be performed by the creator of the portfolio or by an admin. The options to manage portfolios will not appear in the user interface if these conditions are not met, and these permissions are also checked on the backend.

4.6.1 Rename Portfolio

“Rename” functionality is accessed through the portfolio options menu on the upper-right corner of the portfolio.

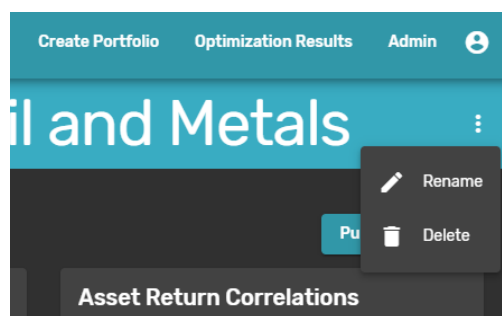


Figure 29: Portfolio option menu

After “Rename” is selected, the “Rename Portfolio” dialog will appear, where users can rename and save or cancel.

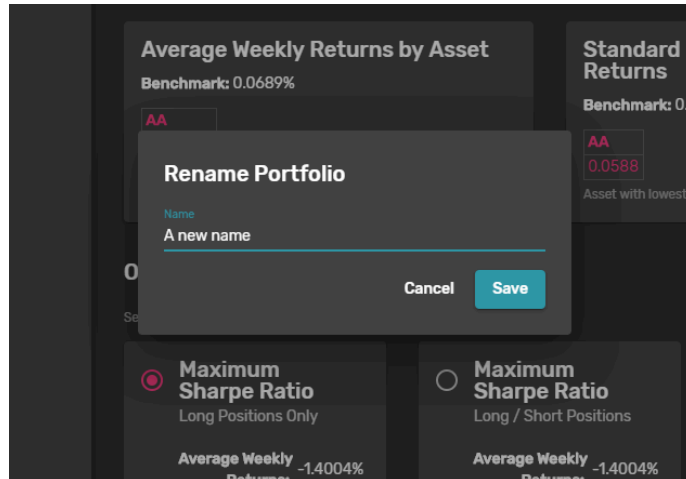


Figure 30: Rename Portfolio dialog

4.6.2 Delete Portfolio

“Delete” functionality is accessed through the portfolio options menu on the upper-right corner of the portfolio. After “Delete” is selected, a dialog will appear asking for confirmation to delete the portfolio.

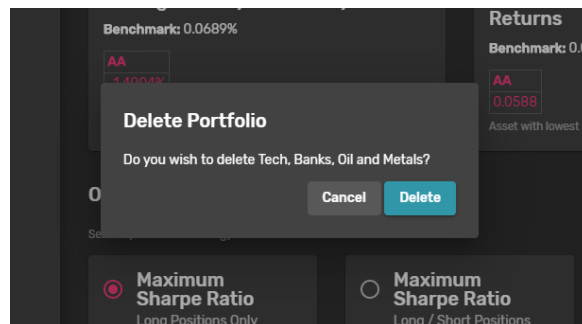


Figure 31: Delete Portfolio dialog

If the portfolio is deleted, the user will be navigated to the “Select a Portfolio” route, and the portfolio will be removed from the navigation list.

4.6.3 Publish Portfolio

“Publish Portfolio” functionality is accessed through the button by the same name in the upper-right section of unpublished portfolios.

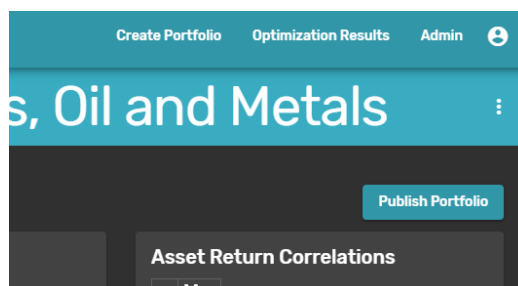


Figure 32: Publish Portfolio button

Once a portfolio is published, this button will disappear and will be replaced by a published confirmation.

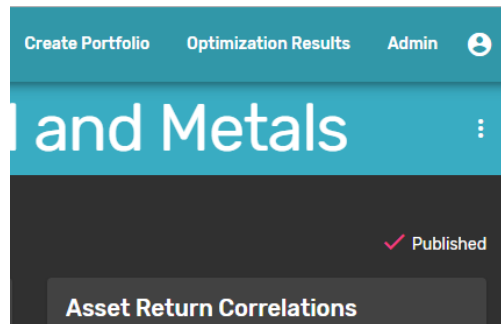


Figure 33: Published confirmation

If a user publishes a portfolio, it will remain in the “My Portfolios” list for that user, but it will appear in the “Community Portfolios” list for other users.

5. OPTIMIZATION METHODOLOGY

The methodology for performing the portfolio optimization in the application was influenced by Professor Colby Wright of Brigham Young University [3]. He demonstrates portfolio optimization in Microsoft Excel, using built in Excel functions and the Excel solver add-on. Portfolio Optimizer uses Professor Wright’s approach as a guideline but implements it in Python using NumPy for the mathematics and SciPy for the optimization. The process for performing optimization explained in both Excel and Python below:

1. Gather prices of several financial assets over time.

	A	B	C	D	E	F	G	H	I
1		July 2006 - July 2012 Monthly Prices							
2	Date	IVV	GLD	QQQ	XLU	XLF	IHE	VIG	LQD
3	7/2/2012	\$135.95	\$153.71	\$64.12	\$36.83	\$14.47	\$87.05	\$56.67	\$118.52
4	6/1/2012	\$136.75	\$155.19	\$64.16	\$36.99	\$14.64	\$86.61	\$56.69	\$117.20
5	5/1/2012	\$131.23	\$151.62	\$61.92	\$35.53	\$13.94	\$80.84	\$55.48	\$116.19
6	4/2/2012	\$139.62	\$161.88	\$66.61	\$35.33	\$15.36	\$83.64	\$58.02	\$115.31

Figure 34: Asset prices in Excel (Not all data shown)

2. Calculate the log returns of all assets. Most financial models use natural logarithmic returns over arithmetic / discrete returns.

Excel (Not all data shown), using $=LN(B3/B4)$, with row numbers incrementing:

J	K	L	M	N	O	P	Q
Compounded Monthly Returns							
IVV	GLD	QQQ	XLU	XLF	IHE	VIG	LQD
-0.59%	-0.96%	-0.06%	-0.43%	-1.17%	0.51%	-0.04%	1.12%
4.12%	2.33%	3.55%	4.03%	4.90%	6.89%	2.16%	0.87%
-6.20%	-6.55%	-7.30%	0.56%	-9.70%	-3.40%	-4.48%	0.76%
-0.66%	-0.15%	-1.18%	1.74%	-2.32%	1.07%	-0.34%	1.07%

Figure 35: Asset returns in Excel (Not all data shown)

Portfolio Optimizer, AssetData class in backend\server\optimizer\prep_data.py:

```
def generate_returns(prices: List[float]) -> List[float]:
    """Generates a list of rates of returns using natural log
    from a list of asset prices

    :param prices: List[float]
    :return:
    :rtype: List[float]
    """
    ret_val = []
    for i in range(0, len(prices) - 1):
        ret_val.append(np.log(prices[i] / prices[i + 1]))
    return ret_val
```

Figure 36: Asset return calculations in Python

3. Calculate the average and standard deviation of the log returns.

Excel, using *AVERAGE* and *STDEV* formulas:

Average:	0.21%	1.24%	0.81%	0.39%	-0.97%	0.87%	0.36%	0.60%
Std Dev:	5.22%	5.86%	6.15%	4.19%	8.87%	4.61%	4.37%	2.67%

Figure 37: Asset return averages and standard deviation in Excel

Portfolio Optimizer, AssetData class in backend\server\optimizer\prep_data.py:

```
self.std_dev = np.std(self.returns)
self.avg_return = np.average(self.returns)
```

Figure 38: Standard deviation and average returns in Python

4. Calculate excess returns of all assets, defined as log returns minus the average return of the asset.
5. Generate the Variance-Covariance matrix, defined as

$$\left(\frac{1}{n}\right)X^T X$$

where X is the excess returns of all assets. This is done over two steps in both approaches:

Excel (calculating $X^T X$), using $=MMULT(TRANSPOSE(Data!R3:Y74),Data!R3:Y74)$, where $Data!R3:Y74$ is the cells containing the excess return values:

12		IVV	GLD	QQQ	XLU	XLF	IHE	VIG	LQD
13	IVV	0.193541	0.026191	0.209972	0.099878	0.291198	0.133852	0.157683	0.034756
14	GLD	0.026191	0.243922	0.012981	0.018673	0.010249	0.034734	0.025409	0.018985
15	QQQ	0.209972	0.012981	0.268467	0.110159	0.287038	0.135793	0.163192	0.03828
16	XLU	0.099878	0.018673	0.110159	0.124591	0.106753	0.087588	0.084173	0.03232
17	XLF	0.291198	0.010249	0.287038	0.106753	0.55856	0.191858	0.236358	0.053719
18	IHE	0.133852	0.034734	0.135793	0.087588	0.191858	0.150934	0.11419	0.036432
19	VIG	0.157683	0.025409	0.163192	0.084173	0.236358	0.11419	0.13543	0.029285
20	LQD	0.034756	0.018985	0.03828	0.03232	0.053719	0.036432	0.029285	0.050464

Figure 39: $X^T X$ Matrix in Excel

Portfolio Optimizer, AssetMatrices class in backend\server\optimizer\prep_data.py:

```
def generate_x_transpose_x_matrix(self) -> np.ndarray:
    return np.matmul(self.excess_returns_matrix, self.excess_returns_matrix.T)
```

Figure 40: $X^T X$ Matrix generation in Python

Excel (dividing previous matrix by n), using $=B13:I20/F2$, where $B12:I20$ is the previous matrix and $F2$ is n :

24		IVV	GLD	QQQ	XLU	XLF	IHE	VIG	LQD
25	IVV	0.002726	0.000369	0.002957	0.001407	0.004101	0.001885	0.002221	0.00049
26	GLD	0.000369	0.003436	0.000183	0.000263	0.000144	0.000489	0.000358	0.000267
27	QQQ	0.002957	0.000183	0.003781	0.001552	0.004043	0.001913	0.002298	0.000539
28	XLU	0.001407	0.000263	0.001552	0.001755	0.001504	0.001234	0.001186	0.000455
29	XLF	0.004101	0.000144	0.004043	0.001504	0.007867	0.002702	0.003329	0.000757
30	IHE	0.001885	0.000489	0.001913	0.001234	0.002702	0.002126	0.001608	0.000513
31	VIG	0.002221	0.000358	0.002298	0.001186	0.003329	0.001608	0.001907	0.000412
32	LQD	0.00049	0.000267	0.000539	0.000455	0.000757	0.000513	0.000412	0.000711

Figure 41: Variance-Covariance Matrix in Excel

Portfolio Optimizer, AssetMatrices class in backend\server\optimizer\prep_data.py:

```
def generate_variance_covariance_matrix(self) -> np.matrix:
    return self.x_transpose_x_matrix / self.n
```

Figure 42: Variance-Covariance Matrix generation in Python

6. Generate the Standard Deviation Matrix, defined as dd^T , where d is a vector of the standard deviation of returns of the assets.

Excel, using $=MMULT(B2:B9, TRANSPOSE(B2:B9))$, where $B2:B9$ is the vector of standard deviation of returns:

35		IVV	GLD	QQQ	XLU	XLF	IHE	VIG	LQD
36	IVV	0.002726	0.00306	0.003211	0.002187	0.004631	0.002407	0.00228	0.001392
37	GLD	0.00306	0.003436	0.003604	0.002455	0.005199	0.002702	0.00256	0.001563
38	QQQ	0.003211	0.003604	0.003781	0.002576	0.005454	0.002835	0.002686	0.001639
39	XLU	0.002187	0.002455	0.002576	0.001755	0.003716	0.001931	0.00183	0.001117
40	XLF	0.004631	0.005199	0.005454	0.003716	0.007867	0.004089	0.003874	0.002365
41	IHE	0.002407	0.002702	0.002835	0.001931	0.004089	0.002126	0.002014	0.001229
42	VIG	0.00228	0.00256	0.002686	0.00183	0.003874	0.002014	0.001907	0.001164
43	LQD	0.001392	0.001563	0.001639	0.001117	0.002365	0.001229	0.001164	0.000711

Figure 43: Standard Deviation Matrix in Excel

Portfolio Optimizer, AssetMatrices class in backend\server\optimizer\prep_data.py:

```
def generate_std_dev_matrix(self) -> np.ndarray:
    return np.outer(self.std_dev_vec, self.std_dev_vec.T)
```

Figure 44: Standard Deviation Matrix generation in Python

7. Generate the Correlation Matrix by dividing the Variance-Covariance Matrix by the Standard Deviation Matrix.

Excel, using $=B25:I32/B36:I43$, where $B25:I32$ is the Variance-Covariance Matrix and $B36:I43$ is the Standard Deviation Matrix:

46		IVV	GLD	QQQ	XLU	XLF	IHE	VIG	LQD
47	IVV	1	0.120544	0.921148	0.64319	0.885659	0.783153	0.973956	0.351679
48	GLD	0.120544	1	0.050728	0.107113	0.027766	0.181024	0.139797	0.171115
49	QQQ	0.921148	0.050728	1	0.602325	0.741241	0.67459	0.855847	0.328876
50	XLU	0.64319	0.107113	0.602325	1	0.40467	0.63872	0.647992	0.407602
51	XLF	0.885659	0.027766	0.741241	0.40467	1	0.660771	0.859363	0.319964
52	IHE	0.783153	0.181024	0.67459	0.63872	0.660771	1	0.798687	0.417443
53	VIG	0.973956	0.139797	0.855847	0.647992	0.859363	0.798687	1	0.354243
54	LQD	0.351679	0.171115	0.328876	0.407602	0.319964	0.417443	0.354243	1

Figure 45: Correlation Matrix in Excel

Portfolio Optimizer, AssetMatrices class in backend\server\optimizer\prep_data.py:

```
def generate_correlation_matrix(self) -> np.ndarray:
    return np.divide(self.variance_covariance_matrix, self.std_dev_matrix)
```

Figure 46: Correlation Matrix generation in Python

8. Generate a section where the optimization results can be displayed, add cells to prepare for optimization

13		Equal Weight	Max Return	Min St Dev	Max Sharpe	Max Sharpe	Max Ret	Min St Dev	Max Ret
14	Constraint	<i>None</i>	2.67%	1.24%	<i>None</i>	<i>No Short</i>	<i>Short OK</i>	<i>No Short</i>	<i>No std dev</i>
15									
16	IVV	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
17	GLD	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
18	QQQ	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
19	XLU	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
20	XLF	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
21	IHE	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
22	VIG	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
23	LQD	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
24	T. Weight	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
25	Return	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
26	Std Dev	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
27	Sharpe	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!

Figure 47: Optimization results section in Excel, before optimization

Note: the constraint for the “Max Return” strategy is the lowest standard deviation of all assets, and the constrain for the “Min St Dev” strategy is the highest average return for all assets.

9. Calculate total weights by summing the weights of all assets.
10. Calculate returns of each portfolio by matrix multiplying the portfolio weights against the average returns vector.

Excel Formula: $=MMULT(TRANSPOSE(B16:B23),\$C\$2:\$C\$9)$, where $B16:B23$ is the portfolio weights and $\$C\$2:\$C\9 is the average returns vector.

Portfolio Optimizer, Optimize class in backend\server\optimizer\optimize.py:

```
@staticmethod
def calculate_returns(weights_vec: np.ndarray, avg_returns_vec: np.ndarray) ->
np.ndarray:
    """Dot product of weight and return vectors

    :param weights_vec: np.ndarray
    :param avg_returns_vec: np.ndarray
    :return:
    """
    return np.dot(weights_vec, avg_returns_vec)
```

Figure 48: Portfolio returns calculation in Python

11. Calculate the portfolio standard deviation.

Excel Formula:

$=SQRT(MMULT(MMULT(TRANSPOSE(B16:B23),\$F\$3:\$M\$10),B16:B23))$, where $B16:B23$ is the portfolio weights and $\$F\$3:\$M\10 is the Variance Covariance Matrix

Portfolio Optimizer, Optimize class in backend\server\optimizer\optimize.py:

```
@staticmethod
def calculate_std_dev(weights_vec: np.ndarray, variance_covariance_matrix:
np.ndarray) -> float:
    """Square root of 1 x 1 matrix of standard deviations

    :param weights_vec:
    :param variance_covariance_matrix:
    :return:
    """
    inner_matrix = np.matmul(weights_vec, variance_covariance_matrix)
    outer_matrix = np.matmul(inner_matrix, weights_vec)
    return sqrt(outer_matrix)
```

Figure 49: Portfolio return standard deviation calculation in Python

12. Calculate the portfolio's Sharpe Ratio. The standard formula for the Sharpe Ratio is

$$\frac{R_p - R_f}{\sigma_p}$$

where R_p is the returns of the portfolio, R_f is the risk free rate, and σ_p is the standard deviation of returns of the portfolio. Professor Wright's demonstration and Portfolio Optimizer both use a simplified Sharpe Ratio where the risk free rate is omitted.

13. Optimize the portfolio, taking constraints into consideration. It is also possible for the optimizer to allow short selling, that is, where people make money if an asset goes down in value by "short selling" it. In these cases, asset weights may be negative.

Excel (Using Excel Solver):

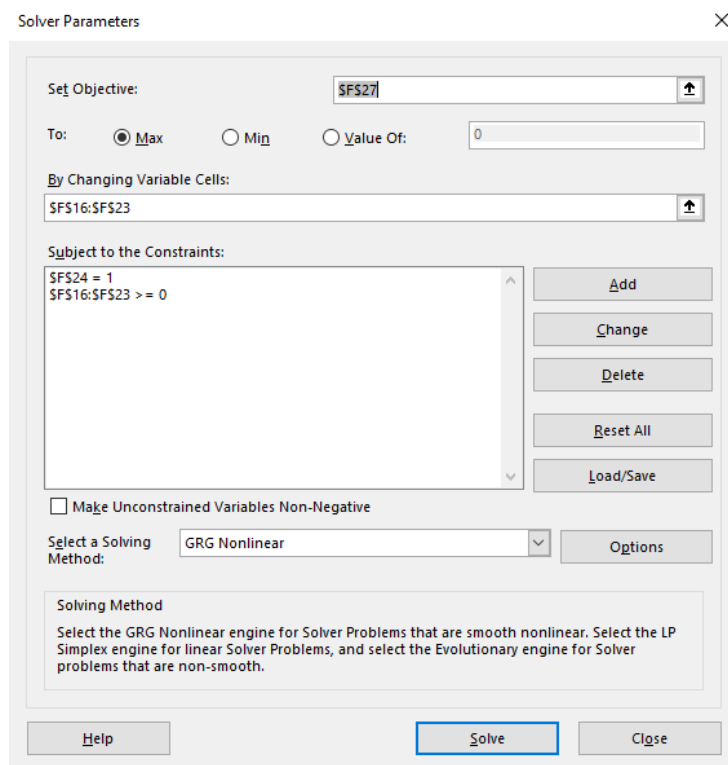


Figure 50: Example Excel Solver optimization configuration

This is an example of the solver configuration for the “Max Sharpe Ratio, No Short Selling” strategy. The solver will adjust the weights until the $F27$ (where the Sharpe Ratio is calculated) is maximized all weights must be positive (because no short selling is allowed), and the sum of all weights must equal 1 (calculated in cell $F24$.)

Portfolio Optimizer, Optimize class in backend\server\optimizer\optimize.py:

There are three optimization methods in this class. They all utilize SciPy > Optimize > Minimize with the SLSQP algorithm. Each method passes a different optimization function to achieve its objective. Each method may also pass different constraints and bounds.

Every optimizer uses a “the sum of assets weights must equal one” constraint:

```
weights_equal_1_constraint = {
    'type': 'eq',
    'fun': lambda arr: reduce(lambda acc, cur: acc + cur, arr) - 1
}
```

Figure 51: Python optimizer “weights must equal 1” constraint

Each optimizer may also pass long only or long / short bounds:


```

# Bounds for optimizer must match length of data
self.long_only_bnds = [[0, 1] for x in asset_matrices.asset_data]
self.short_ok_bnds = [[-1, 1] for x in asset_matrices.asset_data]

```

Figure 52: Python optimizer bounds

Example optimization function:

```

def generate_max_sharpe_ratio(self, shorting_allowed=False) -> OptimizeOutcome:
    def max_sharpe_fn(weights_vec: np.ndarray):
        ret = self.calculate_returns(weights_vec, self.asset_matrices.avg_returns_vec)
        std_dev = self.calculate_std_dev(weights_vec, self.asset_matrices.variance_covariance_matrix)
        return (ret / std_dev) * -1

    bnds = self.short_ok_bnds if shorting_allowed else self.long_only_bnds

    optimize_result: OptimizeResult = minimize(
        max_sharpe_fn,
        self.equal_weights,
        method='SLSQP',
        bounds=bnds,
        constraints=[self.weights_equal_1_constraint]
    )

    pw = self.process_weights(optimize_result.x)
    portfolio_returns = PortfolioReturns(self.asset_matrices.asset_data, optimize_result.x)

    return OptimizeOutcome(OptimizeGoal.MAX_SHARPE, shorting_allowed, optimize_result.x,
        pw['returns'], pw['std_dev'], pw['sharpe_ratio'], optimize_result, portfolio_returns)

```

Figure 53: Example optimizer function from Portfolio Optimizer

6. BENCHMARK COMPARISON METHODOLOGY

All portfolios require a benchmark index as a parameter. This is because all optimization strategies calculate multiple statistics of the optimized portfolio's performance relative to the benchmark index's performance.

6.1 BETA

Beta is a measure of volatility of a portfolio against a benchmark index, usually a benchmark index that represents overall market returns. It defined as

$$\beta = \frac{\sigma_{R_p R_m}}{\sigma_{R_p}}$$

where R_p are the returns of the portfolio and R_m are the returns of the market [5].

In Portfolio Optimizer, is implemented using SciPy's "stats" module (as its "lineregress" function performs the same calculation):

```
def calculate_beta(p_returns: List[float], b_returns: List[float]) -> float:
    slope, *_ = stats.linregress(b_returns, p_returns)
    return float(slope)
```

Figure 54: Portfolio beta calculation

6.2 CAPITAL ASSET PRICING MODEL (CAPM)

CAPM is a way to assess the expected returns of a portfolio, considering the beta (volatility) of the portfolio relative to the market and the time value of money, as determined by the risk free rate. It is defined as

$$E(R_p) = R_f + \beta_p(R_m - R_f)$$

where β_p is the beta of the portfolio, R_f is the risk-free rate, and R_m is the realized return of the market/benchmark index [6]. CAPM is implemented in the application as:

```
def calculate_capm(beta: float, b_tot_ret: float, risk_free_ret=0) -> float:
    return risk_free_ret + beta * (b_tot_ret - risk_free_ret)
```

Figure 55: Portfolio CAPM calculation

6.3 JENSEN'S ALPHA

Alpha is a risk-adjusted measurement of a portfolio's performance relative to its expected returns (as defined by CAPM.) Since it is adjusted for risk, it is a popular measurement for the quality of a portfolio. It is defined as

$$\alpha = R_p - [R_f + \beta_p(R_m - R_f)]$$

where R_p is the realized return of the portfolio, β_p is the beta of the portfolio, R_f is the risk-free rate, and R_m is the realized return of the market/benchmark index [7]. It is implemented in the application as:

```
def calculate_alpha(p_tot_ret: float, capm: float) -> float:
    return p_tot_ret - capm
```

Figure 56: Portfolio alpha calculation

7. PROJECT MANAGEMENT PROCESS

This project was developed using a simplified agile development process. I used Git, GitHub, GitHub project management tools, and Travis CI to assist with this process.

7.1 ISSUES

I broke up the pieces of development into manageable issues; I aimed for each issue to take 2 hours to 2 days. The issues can be seen here:

<https://github.com/jtucke2/Portfolio-Optimizer/issues>

An example of an issue is “Portfolio Rename & Delete #46”, where the scope of the issue was to add API routes to rename and delete portfolios, add a service in the UI to communicate with these routes, and add UI components to carry out these tasks.

7.2 SPRINTS

To assist in time management during the project, I broke up the tasks into 3-4 weeklong sprints. and I used GitHub “Milestones” to group issues together into sprints/phases:

<https://github.com/jtucke2/Portfolio-Optimizer/milestones>

7.3 KANBAN BOARD

To assist with organization of task progress and competition, I created a Kanban board. I used a GitHub project with “Todo,” “In Progress,” and “Done” columns:

<https://github.com/jtucke2/Portfolio-Optimizer/projects/1>

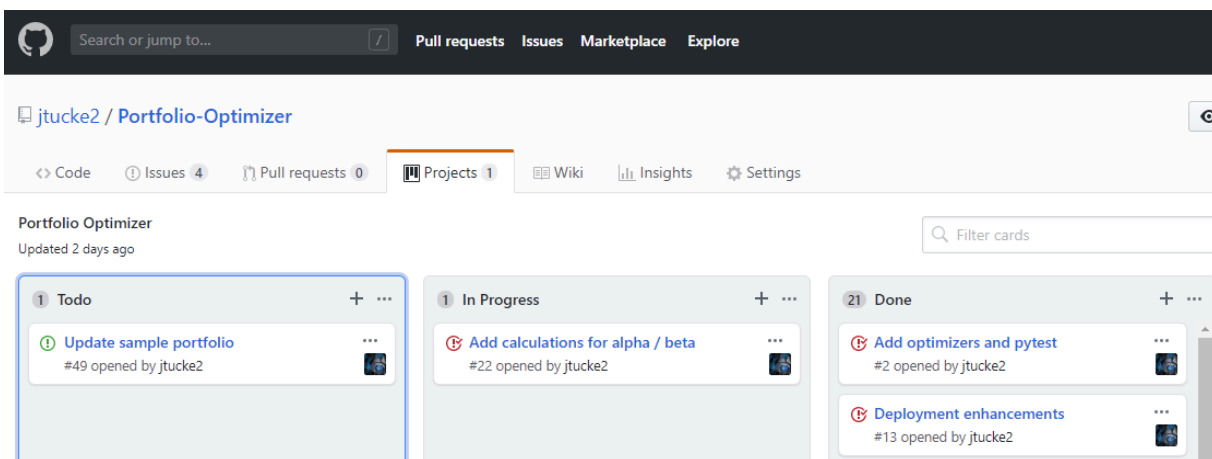


Figure 57: Portfolio Optimizer GitHub project board

7.4 GIT PROCESS AND CONTINUOUS INTEGRATION

I used a process inspired by GitFlow to manage my source code repository. GitFlow suggests completing each feature on a separate branch then merging the features branch into the “develop” branch when the feature is complete [8]. I created a new branch for each issue, which was to be merged into the develop branch when the issue was complete.

When writing Git commit messages, I used GitHub specific issue references, such as including “#46” in the commit message so the commit would be automatically referenced on the issue.

Portfolio rename & delete #46

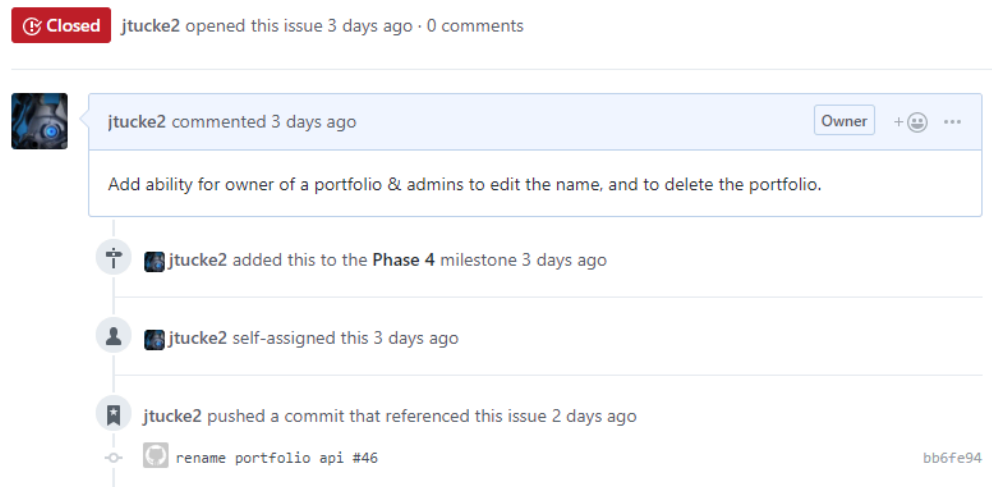


Figure 58: Automatic commit reference on issue page

When the issue was complete, I could include a GitHub issue closing reference phrase, which would automatically close the issue when it was, such as “fixes #42” [9]. When issues were complete, I put in a pull request for the issue to be merged into develop.

I configured a cloud-based continuous integration tool, Travis CI, to run and verify pull requests. This tool was configured to build my project, run TypeScript and CSS lint on the front end, and run pycodestyle on the backend. If any of these steps failed, the PR could not be merged.

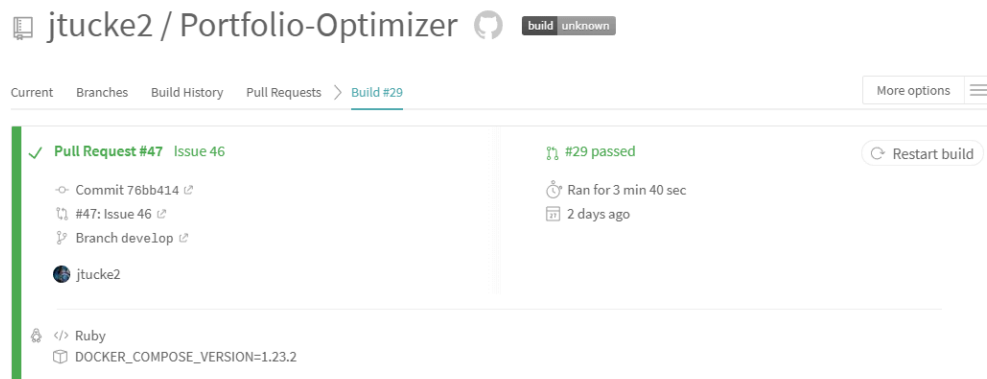


Figure 59: Example Travis CI build report

Every week, I would merge develop into the master branch so it would have up to date code.

8. CHALLENGES ENCOUNTERED AND LESSONS LEARNED

During my time working on this project, I encountered multiple challenges and learned some lessons. Some of these challenges were self-imposed, while some were unexpected.

I purposefully challenged myself to choose a project with higher mathematics complexity than the programming I normally perform. I have never taken a linear algebra class, and the highest-level mathematics class I completed as a Business Administration major was Business Calculus. Recognizing its importance to programming, I have been sporadically studying linear algebra for the past couple years and I wanted to put those concepts to use. Through this project, I was able to learn a lot about implementing linear algebra in NumPy.

I also chose to incorporate some technologies that I was not familiar with. Coworkers of mine have long recommended the Python package Celery, but I have never used Celery, RabbitMQ, or any task queue for that matter. Learning Celery and RabbitMQ was generally a smooth and worthwhile experience for this project.

I first found Professor Wright's portfolio optimization spreadsheet in 2012, and I started making portfolio optimization spreadsheets back then. 2012 was before I learned to program. Ever since I have learned to program, I have always been curious about implementing something equivalent to the Excel Solver, and this project prompted me to learn SciPy to do that. Overall, it took me longer to convert the core of the portfolio optimizer spreadsheet to Python than I thought it would. Converting the linear algebra from Excel to NumPy brought with it several small hitches. Converting the optimizer from Excel Solver to SciPy was more difficult and that took a while.

One unexpected challenge was collecting historical financial data. Many free finance APIs do not have long historical data, and generally this took significantly more time I thought it would. The solution I settled on was a Python package that would scrap data from Yahoo Finance, yahoofinancials, however this is a poorly coded package that leaves a lot to be desired. For the sake for time, I stuck with this package and that will bring some undesirable behavior to my application. Additionally, I prefer that my application is easy to install (currently it takes 2 commands if Docker is installed), so that is another reason I stuck with yahoofinancials, which requires no API key.

I work professionally as a full stack software developer am well experienced with Angular, Nginx, MongoDB, and Docker, but that does not mean the challenges I faced with these technologies were easy to deal with. I also tried to implement some lessons learned some previous projects into this one, and I felt I was able to do that with some improvements to the Angular and Docker configurations.

While I consider myself moderately proficient with Python, it is not my most proficient backend language. I felt like I learned a lot about Python by doing this project. I used pytest unit testing in the project to test certain classes as I was developing them, but I found that keeping the tests

working was too brittle and not a wise use of time, so the unit tests are not currently in a passing state.

One notable lesson learned was time management doing a complex project while working. I have not coded a project of this scope outside of the workplace before, and some tasks that I thought were quick/easy to do on project at work seemed to take long on this project. I generally felt like I under estimated the time it would take to complete many tasks. I attribute that to the fact that work is 8 hours a day, so if something takes 2 days to do at work would take a week to do at home.

9. THE FUTURE FOR PORTFOLIO OPTIMIZER

I plan on continuing development of the project after the class ends and I have already been writing issues on the project page intended for after the class. The project is open source, and I hope one day that other people find the project and become users and/or contributors. The project can be found here: <https://github.com/jtucke2/Portfolio-Optimizer>

Some plans for the project include:

- Upgrade to Angular version 8 when it releases this summer
- Add Alpha optimization Strategy
- Fix the unit tests in Python backend, add unit tests in Angular frontend
- Add unit testing of the frontend and backend to the Travis CI job
- Add TSDocs to the frontend and Sphinx docs to the backend
- Add a docker-compose file that will generate and host HTML files for unit test coverage reports and code documentation
 - Code alternative solution to retrieve assets prices to replace yahoofinancials

10. CONCLUSION

This project was challenging, and more time consuming than I anticipated. I felt like multiple classes I have taken at Towson University helped me get through these challenges, including but not limited to AIT 616, AIT 618, COSC 578, and COSC 600. I feel like I come out a better Python program and significantly better at linear algebra. I also was glad to achieve a long-held goal of converting the portfolio optimizer to a user-friendly program.

11. REFERENCES

- [1] M. Hargrave, "Sharpe Ratio Definition", *Investopedia*, 2019. [Online]. Available: <https://www.investopedia.com/terms/s/sharperatio.asp>. [Accessed: 11- May- 2019].
- [2] S. Slade, "Modern Portfolio Theory: A Dialectic Approach", *Yale University*, 2006. [Online]. Available: <http://zoo.cs.yale.edu/classes/cs458/lectures/old/mpt/Modern%20Portfolio%20Theory.html>. [Accessed: 11- May- 2019].
- [3] C. Wright, "Generating the Variance-Covariance Matrix", *YouTube*, 2012. [Video File]. Available: <https://www.youtube.com/watch?v=ZfJW3ol2FbA>. [Accessed: 11- May- 2019].
- [4] Python.org, "GlobalInterpreterLock", *Python Software Foundation*, 2017. [Online]. Available: <https://wiki.python.org/moin/GlobalInterpreterLock>. [Accessed: 11- May- 2019].
- [5] W. Kenton, "Beta Definition", *Investopedia*, 2019. [Online]. Available: <https://www.investopedia.com/terms/b/beta.asp>. [Accessed: 11- May- 2019].
- [6] W. Kenton, "Capital Asset Pricing Model (CAPM)", *Investopedia*, 2019. [Online]. Available: <https://www.investopedia.com/terms/c/capm.asp>. [Accessed: 11- May- 2019].
- [7] J. Chen, "Jensen's Measure", *Investopedia*, 2019. [Online]. Available: <https://www.investopedia.com/terms/j/jensensmeasure.asp>. [Accessed: 11- May- 2019].
- [8] atlassian.com, "Gitflow Workflow", 2019. [Online]. Available: <https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow>. [Accessed: 11- May- 2019].
- [9] github.com, "Closing issues using keywords", 2019. [Online]. Available: <https://help.github.com/en/articles/closing-issues-using-keywords>. [Accessed: 11- May- 2019].