Software Design Specification

for

BASIS Automated Stock Investment Software

Version 1.4 approved

Prepared by Vinayak Mathur and

Anurag Upadhya

Case Western Reserve University

October 8th 2020

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
Vinayak Mathur Anurag Upadhya	10/04/20	First Draft	1.0
Vinayak Mathur Anurag Upadhya	10/05/20	3. Principal Classes and 4. Interagency Protocols Added	1.1 1.2
Anurag Upadhya	10/07/20	Revision	1.3
Vinayak Mathur Anurag Upadhya	10/08/20	Added Inspection Report, Finished Final Draft	1.4

1. Introduction

Purpose

This document describes the following aspects of the BASIS Automated Stock Investment Software:

- Applicable Documents and References
- Principal classes
- Interactions between Components
- Class interfaces
- Handling failures and Exceptions

The purpose of this document is to specify the implementation aspects of stock screening, stock selection and BASIS Automated Stock Investment Software which is a web-based application for automating portfolio management. To insure the partitioning of functionality between the numerous tools used to construct allocation models, an object-oriented architecture is utilized. Such an approach allows the components of BASIS to interact as entities with defined behaviors. As this document contains the design specifications for version 1.0 of BASIS, implementation details are subject to change for future versions. Please refer to the software requirements specification for related details on the required functionality of BASIS [AD-1].

Scope

BASIS is an application that provides stock recommendations for stock portfolios based on answers provided by the client to pre-set questions. Users are paired with portfolios that match their comfort with risk-tolerance without having to go through the hassle of researching investments and avoiding the expensive fees of wealth management firms. A more detailed outlook of the Project Scope is provided in the SRS Documentation [AD-1].

Abbreviations and Definitions

1.1.1 AB-1: BASIS: BASIS Automated Stock Investment Software
1.1.2 AB-2: OOP: Object Oriented Programming
1.1.3 AB-3: OOD: Object Oriented Design
1.1.4 AB-4: CSV: Comma Separated Values file
1.1.5 AB-5: JSON: JavaScript Object Notation file
1.1.6 AB-6: MPA: Markowitz Portfolio Allocator / Allocator
1.1.7 AB-7: UI: User Interface
1.1.8 DE-1: ticker: A ticker symbol or stock symbol is an abbreviation used to uniquely identify publicly traded shares of a particular stock on a particular stock market.

2. Applicable Documents and References

AD-1: Mathur, V., Upadhya, A., BASIS Automated Stock Investment Software Requirements Specification. 21 September, 2020.

3. Principal Classes

This section enumerates the primary classes responsible for the functioning of BASIS. There are seven principal classes, each of which represents a .py file in the Python code. Refer to section 4.1 for a full class diagram.

DTable

The DTable class utilizes a CSV (Comma Separated Values) file to store the stock ticker information like name, description and current market price.

3.1.1 Attributes

3.1.1.1 ticker: str = None.

Stores the name of the ticker whose information is to be retrieved from the CSV.

3.1.1.2 new_ticker: str = None.

Stores the new ticker that will replace the old ticker.

3.1.1.3 new_tags: list = None.

Stores a list of tags associated with the new ticker.

3.1.1.4 old_ticker: str = None.

This is the old ticker that is to be replaced

3.1.2 Methods

3.1.2.1 get(ticker): dict.

Retrieves useful information for the given ticker like historical data, current market price, company name, company description, link, 52H, 52L, etc.

3.1.2.2 add(new_ticker, new_tags): None.

Adds a completely new ticker to the database.

3.1.2.3 remove(old_ticker): None.

Removes a ticker from the database.

3.1.2.4 modify(old_ticker, new_tags): bool.

Replaces the given ticker's tags with new ones. Returns False if the ticker is not present in the database.

3.1.2.5 get_tags(ticker): list.

Returns the tags of the given ticker.

QRange

The Range class lists the range of the acceptable answers to the questions for each instance of the Question class.

3.1.3 Attributes

3.1.3.1 buttons: list = None

This attribute stores the list of buttons or options that a user has for a question.

3.1.3.2 type_range: str = None

This attribute stores the type of range. It can be a slider, buttons or a user entered value.

3.1.4 Methods

3.1.4.1 __init__(self, buttons): None

Default constructor to initialize a QRange object with buttons.

3.1.4.2 __init__(self, i, j): None

Default constructor to initialize a QRange object with a slider.

3.1.4.3 __init__(self): None

Default constructor to initialize a QRange object with a user entered value.

Question

The Range class is used to store questions in the QSet class.

3.1.5 Attributes

3.1.5.1 question: str = None

This attribute stores a question to be asked to the user.

3.1.5.2 ran: QRange = None

This attribute stores the range of the question.

3.1.6 Methods

3.1.6.1 __init__(self, question, ran): None

Default constructor to initialize a Question object with the given question and range.

QSet

The Range class is used to store questions in the QSet class.

3.1.7 Attributes

3.1.7.1 ques: list = None

This attribute stores the list of questions in that question set instance.

3.1.7.2 name: str = None

This attribute stores the name of the question set instance.

3.1.7.3 priority: int = None

This attribute stores the priority of the question set (highest priority will be asked first).

3.1.8 Methods

3.1.8.1 __init__(self, name, priority, ques):

Default constructor to initialize a QSet object with the list of questions, name and priority.

3.1.8.2 __init__(self, name, priority):

Default constructor to initialize an empty QSet object.

QLibrary

The QLibrary is a set of standard questions with subsets containing specific questions for each individual user. Question sets chosen from this library will be asked to the user. Answers to questions will initiate a response asking for answers for further question sets.

3.1.9 Attributes

3.1.9.1 df: pandas.dataframe = pandas.read_json("questions.JSON")

This attribute retrieves questions from a JSON file and stores the questions in a pandas dataframe.

3.1.9.2 qset: QSet = None.

This attribute stores and adds the given QSet.

3.1.9.3 qset_name: str = None.

This attribute stores the name of the QSet to be modified.

3.1.9.4 old_question: str = None.

This attribute stores the old question to be replaced.

3.1.9.5 new_question: str = None.

This attribute stores the new question that will replace old_question.

3.1.9.6 new_range: list = None.

This attribute stores the new range for new_question

3.1.9.7 question: str = None.

Stores the question to look for its range.

3.1.9.8 passkey: str = None.

This attribute stores the passkey required to make permanent changes to the database.

3.1.10 Methods

3.1.10.1 add(self, qset): None.

Add the given question set to the QLibrary.

3.1.10.2 replace(self, qset_name, old_question, new_question, new_range): bool.

Replace the old question from the given qset with the new question. Return True if the question was successfully replaced and False if it wasn't found

3.1.10.3 find(self, question): QRange.

This method returns the range of the given question.

3.1.10.4 display(self): str.

This method returns the question library as a string.

3.1.10.5 remove(self, question): None.

Removes the question from the question library.

3.1.10.6 exists(self, question): bool.

Returns True if the question exists in the library and False if it does not.

3.1.10.7 clear(self, passkey): bool.

If the right passkey is provided then the method will clear the entire library. Returns false if the passkey is incorrect.

3.1.10.8 modify(self, question, new_options): bool.

Replaces the old options for the question with new ones. Return True if the question exists and False if not.

SSelector

The SSelector class selects stocks based on the user's answers to the questions in QLibrary.

3.1.11 Attributes

3.1.11.1 finalCons: list = list() (empty list)

Initialized as an empty list, this variable stores the final conditions selected from the conditions.csv file.

3.1.11.2 df: DataFrame = pandas. read(conditions.csv)

Reads the .csv file containing the conditions of the stocks

3.1.12 Methods

3.1.12.1 __init__(self): int

This prints the index and row of the tag.

Fetcher

Fetcher retrieves stock information based on the tags provided.

3.1.13 Attributes

3.1.13.1 tag: str = None.

Stores the tag for which to find respective stocks.

3.1.14 Methods

3.1.14.1 screen(tag): list.

Returns a list of stocks that contain the given tag.

Allocator

The Allocator class performs stock analysis on the list of selected stocks and uses the modern portfolio theory to get the stocks with the highest expected return for given risk.

3.1.15 Attributes

3.1.15.1 price: list = None

This attribute fetches the price data for stocks based on their ticker and from the start data specified.

3.1.15.2 set_of_tickers, tickers: string = User Input

This is the string containing all the tickers to get prices from. It is a parameter of the function get_price(self, set_of_tickers, start_date)

3.1.15.3 start_date: string: = User Input

This is the string containing the start date. It is of the format 'YYYY-MM-DD'

3.1.15.4 average_return: float = None

This attribute stores the mean of the percentage change in given tickers

3.1.15.5 covariance: float = None

This attribute stores the covariance of the percentage change in the given tickers

3.1.15.6 alpha: float = None

Stores the excess return of the given ticker with respect to the S&P500 index

3.1.15.7 beta: float = None

Stores the volatility of the given ticker with respect to the S&P500 index

3.1.15.8 mean_var: object = None

mean_var computes the mean variance of the returns of the price variable

3.1.15.9 rel: subplot = None

rel is used to plot a graph comparing the relative performance of stocks with each other

3.1.15.10 histogram: ndarray = None

This variable is used to plot the distribution of returns in the form of a histogram

3.1.15.11 hm: module = None

hm is used to plot the heat map visualization of the correlations of stocks

3.1.15.12 stats: NoneType = None

This variable is used to calculate and list statistics of stocks

3.1.16 Methods

3.1.16.1 __init__(self): None

This method is used to initialize the class object

3.1.16.2 get_price(self, set_of_tickers, start_date) : DataFrame

Returns the dates and prices of the set of tickers on the respective dates starting from start_date

3.1.16.3 get_average_return(self, tickers, start_date): float

Computes and returns the average yearly return of tickers from start_date

3.1.16.4 get_covariance(self, tickers, start_date): float

Computes and returns the covariance of tickers from start_date

3.1.16.5 set_portfolio_weights(self, weights): array

Returns the weights as an array of weights

3.1.16.6 get_alpha(self, ticker, start_date): float

This method is used to find the excess return of the ticker with respect to the S&P500 index.

3.1.16.7 get_beta(self, ticker, start_date): float

This method is used to measure the given ticker's volatility with respect to the S&P500 index.

3.1.16.8 portfolio_return(self, tickers, weights): float

Returns the total yearly average return of the portfolio based on past performance taking into account the portfolio weights

3.1.16.9 mean_variance(self, tickers, start_date): float

Returns the mean variance of the given tickers from the given start_date

3.1.16.10 relative_performance_graph(self, tickers, start_date): Subplot

Rebases and plots a graph of given tickers from the given start_date

3.1.16.11 histogram_of_returns(self, tickers, start_date): ndarray

Plots a histogram of returns for each of the given tickers from the given start_date

3.1.16.12 statistics(self, tickers, start_date): NoneType

Displays some general statistics of the given tickers from the given start_date

3.1.16.13 correlations(self, tickers, start_date): DataFrame

Returns the correlations among stocks with given tickers from the given start_date

3.1.16.14 heat_map_visualization(self, tickers, start_date): module

Plots a heat map of the correlations of the given tickers from the given start_date

3.1.16.15 least_risk(self, list): list

This method is used to find stocks with the least amount of risk compared to other stocks with the same return

3.1.16.16 growth(self, list): list

This method is used to find the growth stocks, i.e., the stocks with the high amount of potential return based on previous short-term statistics

3.1.16.17 value(self, list): list

This method is used to find value stocks, i.e., stocks with good fundamentals

Portfolio

The Portfolio class calls functions from the Allocator class to get the stocks with least risk, stocks with highest growth potential, and stocks having great value.

3.1.17 Attributes

3.1.17.1 list: list = UserInput

This is the list of stocks inputted by the user

3.1.17.2 risk_set: set = Allocator.least_risk()

This is the set of stocks with the least amount of risk

3.1.17.3 growth_set: set = Allocator.growth()

This is the set of stocks with the highest potential growth

3.1.17.4 value_set: set = Allocator.value()

This is the set of stocks with the highest value

3.1.17.5 elements1: set = None

This is the set of stocks that are in both risk_set and growth_set, i.e. the intersection of risk_set and growth_set.

3.1.17.6 elements2: set = None

This is the set of stocks that are in both elements1 and value_set, i.e. the intersection of risk_set, growth_set, and value_set.

3.1.18 Methods

3.1.18.1 elements1: set = None

This is the set of stocks that are in both risk_set and growth_set, i.e. the intersection of risk_set and growth_set.

3.1.18.2 __init__(self, list): None

Default constructor to initialize a Portfolio object using the list of stocks provided in the parameter and create allocations based on the given criteria.

3.1.18.3 get_all(self): list

This method returns the entire set of stock tickers in the portfolio to the UI without allocation.

3.1.18.4 get_risk(self): list

This method sends the portfolio with the least risk to the UI.

3.1.18.5 get_value(self): list

This method sends the portfolio with the highest value to the UI.

3.1.18.6 get_growth(self): list

This method sends the portfolio with the highest potential growth to the UI.

3.1.18.7 get_best(self): list

This method sends the portfolio that fits best with the user's provided criteria to the UI.

4. Inter-Agent Protocols

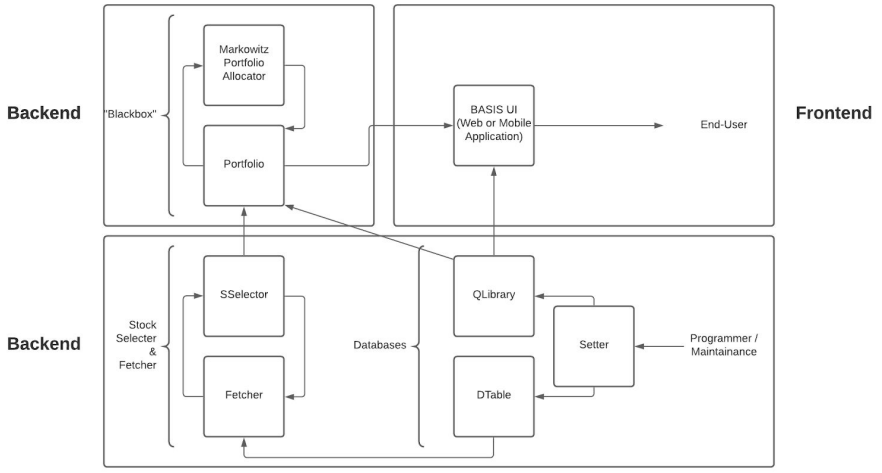
4.1 Interaction Diagrams

The following diagram shows how information is communicated between the different software agents for BASIS's functioning.

Figure 1:

The figure below is a surface overview of the interactions between the four major interfaces of BASIS namely the Databases, the Blackbox, Stock Selector & Fetcher and the Frontend GUI. The interfaces are subdivided into the following principal classes:

1. Databases
    a. DTable
    b. QRange
    c. Question
    d. QSet
    e. QLibrary
2. Blackbox
    a. Portfolio
    b. Markowitz Portfolio Allocator (Allocator)
3. Stock Selector and Fetcher
    a. SSelctor
    b. Fetcher
4. Frontend
    a. Web Application GUI

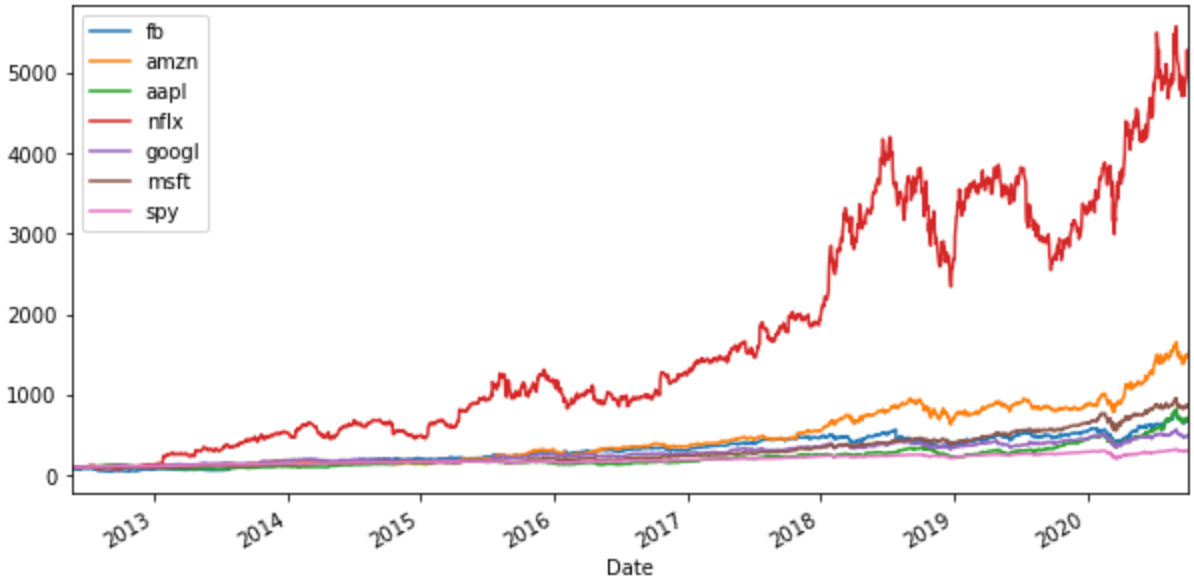


4.2 Stock Selection

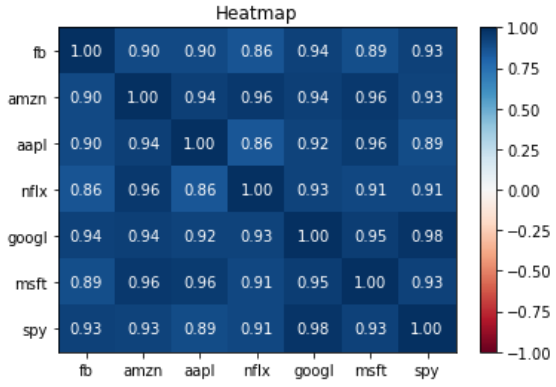
Stock Selection will be conducted by the SelectorFetcher interface. The SSelctor receives a set of criteria noted by the User Interface in response to the questions from the QLibrary that the user answered. This criteria is then analysed by the SSelector to determine a more narrow range of stocks. The SSelector then sends this information to the Fetcher that fetches all the stocks that fit the given criteria.

4.3 Stock Analysis

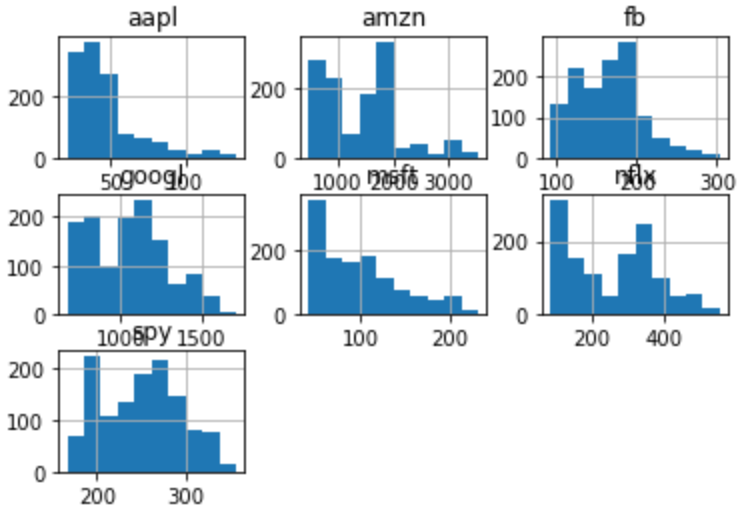
Stock Analysis takes place through the computation of various variables. Mean variance, covariance, average returns, volatility, standard deviation, and risk with respect to the benchmark S&P500 index are the various variables that are computed and evaluated. Graphs are computed for visualization for the backend traders. Some graphs and their explanations are given below.



The above graph shows the relative performance of Facebook (ticker: fb), Amazon (ticker: amzn), Apple (ticker: aapl), Netflix (ticker: nflx), Google (ticker: googl), Microsoft (ticker: msft), and the S&P500 index (ticker: spy). We can see from this that Netflix had the highest growth from 2013 to 2020.



The above heat map shows the correlation among stocks. All stocks have positive correlations indicating that if one of the stocks go up, then the others do as well and vice versa. This is not ideal when constructing a portfolio according to the Markowitz Portfolio Theory as the risk is high.



The above histograms show the distribution of returns for the stocks selected.

Alpha gives us the excess return of a stock with respect to the S&P 500 index. Historically, the S&P 500 index gives an average return of 7-10%. Any return that beats the S&P 500 index is considered a good return. We also take a look at beta, which measures the volatility of a stock with respect to the S&P 500. A beta of less than 1 means that the stock has a low risk, which is lower than that of the S&P 500.

4.4 Portfolio Allocation

In order to allocate the portfolio, we first take the stocks from the SSelector. Next, we use our computations from the stock analysis to label stocks into three categories - least_risk, growth, and value. Stocks in least_risk have a beta value of less than 1. Stocks in growth have an alpha value of greater than 1. Stocks in value have statistics such as the price-to-earnings ratio, earning per share, and revenue over earnings greater than that of the S&P 500 index.

Value stocks are given the greatest portfolio allocations (approximately 0.5 total) as these stocks tend to be less volatile but also have returns close to S&P 500. Stocks in least_risk are given a weight of around 0.25. Even though these stocks are less risky, they tend to have returns lower than those of the S&P 500. Growth stocks are capped at around 0.25 in portfolio allocation weight as they tend to have significant downside risk.

4.5 Portfolio Selection

Depending on how risk-averse the investor is, stocks can be selected on the basis of high risk and high reward, low risk and low reward, or value. If stocks having low risk, high growth, and great value are found, then these stocks can be selected using the get_best() method.

The get_growth() method selects the stocks having both high risk and high reward. The get_risk() method selects the stocking having low risk and low reward. The get_value() method selects stocks that have great value bases on fundamental analysis.

The selected portfolio is sent to the UI when the UI calls a function from the Portfolio class. The class sends the portfolio in JSON format.

4.6 Database Management

Database modules will be managed through the use of passwords and by preventing non-admin users from writing and from reading unless necessary and granted by the admin. This should be achieved by implementing the Setter interface. BASIS contains two primary databases, namely, QLibrary (used to store standard question sets and questions that are to be asked to the user) and DTable (used to store the list of tickers and their data).

4.7 Failures and Exceptions

We will use an inbuilt Python library known as unittest to automate testing to find cases of user input error and unrecognizable user responses. If BASIS is to be implemented in another OOP language, we recommend Selenium or JUnit to be used. BASIS utilizes a basic linter module for Python 3. Small errors will be handled through cases and unrecognizable user responses will give the user a set of valid responses to choose from.

4.8 Web Graphical User Interface

The website shall be built on HTML5 and CSS3 using a Bootstrap 4 framework. BASIS shall implement a Django framework to better work with the Python based backend.

4.9 Plans for Extending the Software

Data structures and algorithms will be used in an attempt to reduce both the Big-O runtime and memory requirements. Memory allocation will also be optimized in an attempt to use as low of memory as possible without decreasing the functionality of BASIS.

4.10 Communication Mechanisms

Communication from the user to the system as well as from the system to the user will take place completely textually. The system will notify the user through a message on the webpage if the system doesn't understand what the user is trying to communicate. The user will enter answers to the questions provided through the UI (webpage). Responses will be made through messages delivered by the UI to the user.

5. Class Interfaces

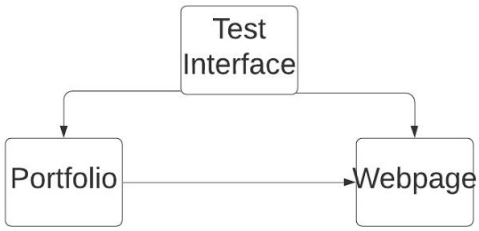
Since we are implementing an OOP language that does not require explicit class abstraction (Python), there is no requirement for separate class interfaces. However, in the event that another OOP language is implemented that does require explicit abstraction (C like languages such as Java, Kotlin, C/C++, C#, etc), we recommend implementing the following class interfaces.

5.1 Setter

The Setter interface is used to modify the database consisting of the QLibrary and the DTable. The purpose of this interface is to prevent non-admins to interact with the database and for admins to have a quick and effortless way to make appropriate changes to the databases without having to access the CSV and JSON files themselves.

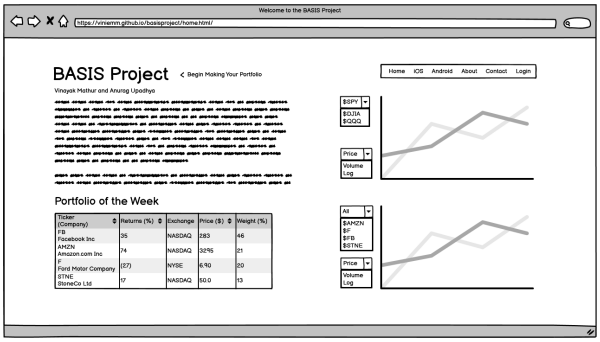
5.2 Test Interface

The test interface is the interface where testing of the Portfolio interactions with the UI will take place. The interface will take a list of stocks as input and give a JSON file as output.



5.3 Graphical User Interface (GUI)

The GUI will utilize the Bootstrap4 CSS framework over HTML5 and Javascript. The purpose of the web GUI will be to enable easy access of the admin protocols from any device that can access the internet. This will also enable users to access BASIS anywhere on the globe.
Figure 5.3.1: BASIS Home Page on a desktop Web Browser.



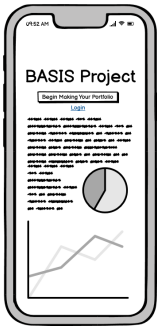
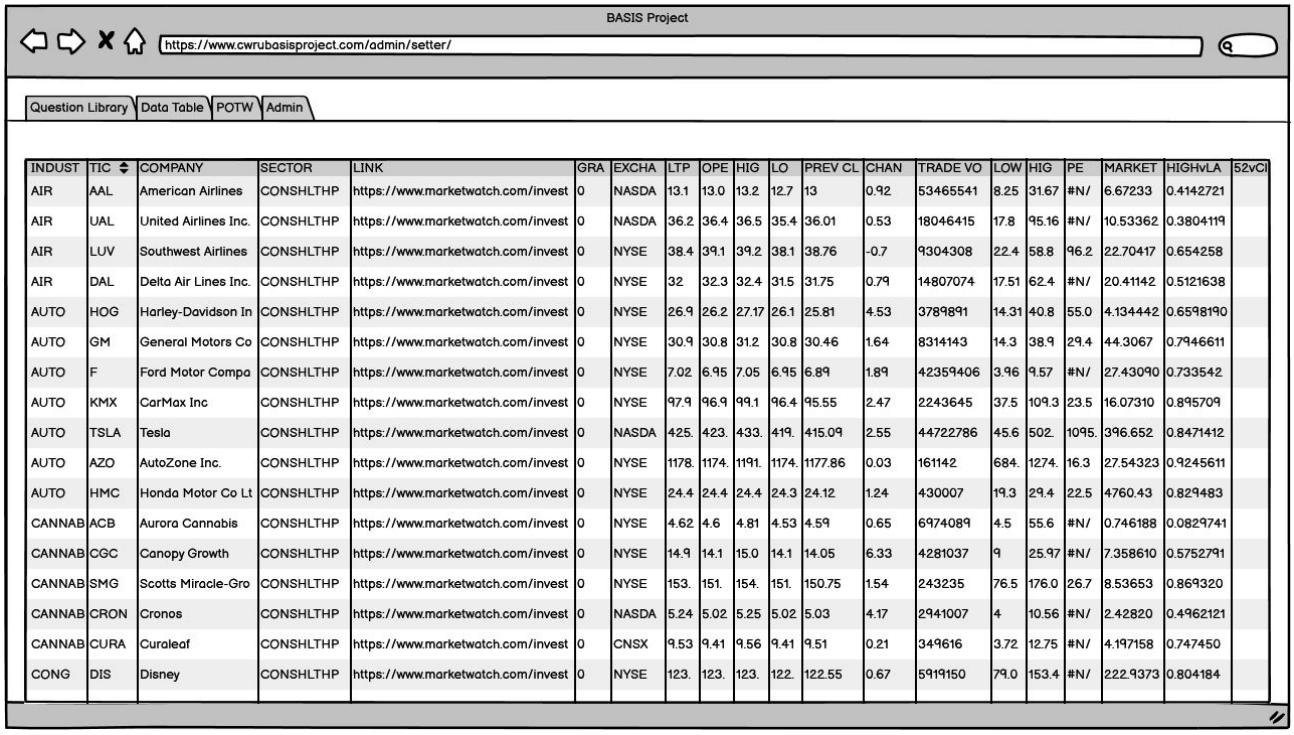


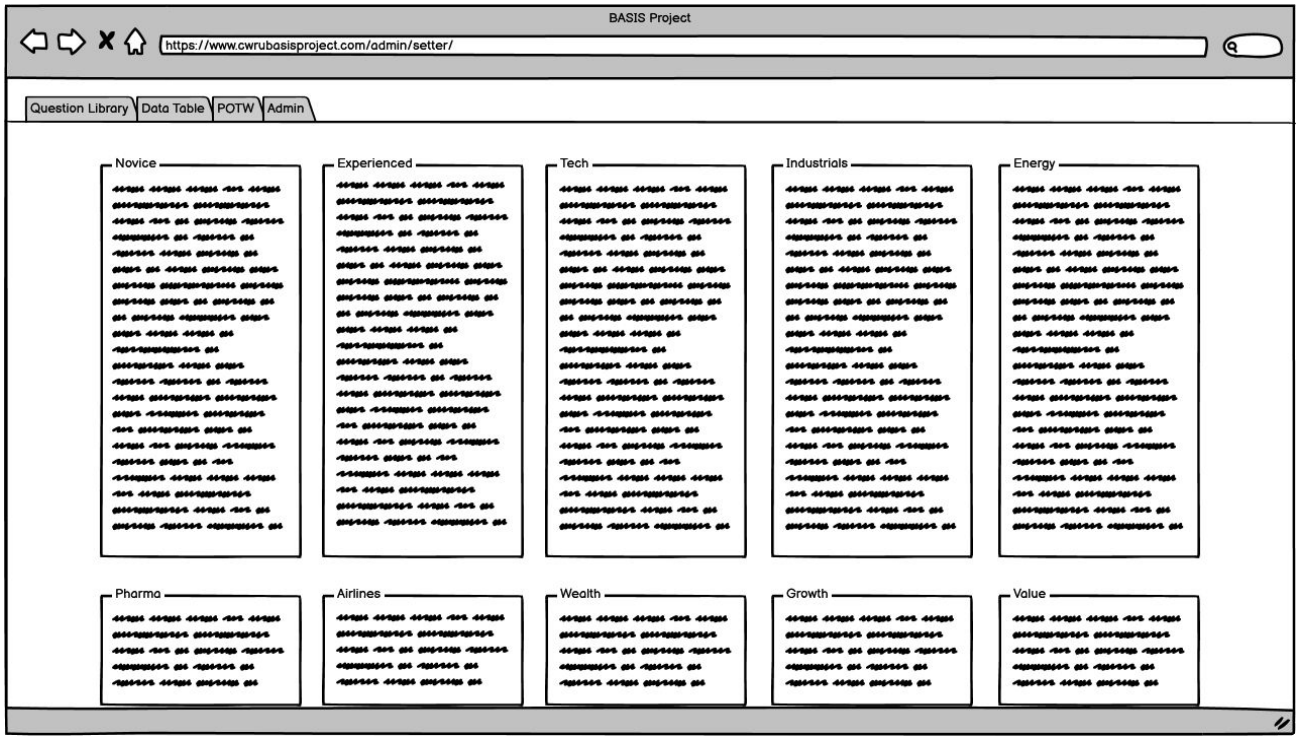
Figure 5.3.2: BASIS Home Page on a Mobile Web Browser

Figure 5.3.3: BASIS's Setter interface showing the Data Table.



INDUST	TIC	COMPANY	SECTOR	LINK	GRA	EXCHA	LTP	OPE	HIG	LO	PREV CL	CHAN	TRADE VO	LOW	HIG	PE	MARKET	HIGHvLA	52vCl
AIR	AAL	American Airlines	CONSHLTHP	https://www.marketwatch.com/invest	0	NASDA	13.1	13.0	13.2	12.7	13	0.92	53465541	8.25	31.67	#N/	6.67233	0.4142721	
AIR	UAL	United Airlines Inc.	CONSHLTHP	https://www.marketwatch.com/invest	0	NASDA	36.2	36.4	36.5	35.4	36.01	0.53	18046415	17.8	95.16	#N/	10.53362	0.3804119	
AIR	LUV	Southwest Airlines	CONSHLTHP	https://www.marketwatch.com/invest	0	NYSE	38.4	39.1	39.2	38.1	38.76	-0.7	9304308	22.4	58.8	96.2	22.70417	0.654258	
AIR	DAL	Delta Air Lines Inc.	CONSHLTHP	https://www.marketwatch.com/invest	0	NYSE	32	32.3	32.4	31.5	31.75	0.79	14807074	17.51	62.4	#N/	20.41142	0.5121638	
AUTO	HOG	Harley-Davidson In	CONSHLTHP	https://www.marketwatch.com/invest	0	NYSE	26.9	26.2	27.17	26.1	25.81	4.53	3789891	14.31	40.8	55.0	4.134442	0.6598190	
AUTO	GM	General Motors Co	CONSHLTHP	https://www.marketwatch.com/invest	0	NYSE	30.9	30.8	31.2	30.8	30.46	1.64	8314143	14.3	38.9	29.4	44.3067	0.7946611	
AUTO	F	Ford Motor Compa	CONSHLTHP	https://www.marketwatch.com/invest	0	NYSE	7.02	6.95	7.05	6.95	6.89	1.89	42359406	3.96	9.57	#N/	27.43090	0.733542	
AUTO	KMX	CarMax Inc	CONSHLTHP	https://www.marketwatch.com/invest	0	NYSE	97.9	96.9	99.1	96.4	95.55	2.47	2243645	37.5	109.3	23.5	16.07310	0.895709	
AUTO	TSLA	Tesla	CONSHLTHP	https://www.marketwatch.com/invest	0	NASDA	425.	423.	433.	419.	415.09	2.55	44722786	45.6	502.	1095.	396.652	0.8471412	
AUTO	AZO	AutoZone Inc.	CONSHLTHP	https://www.marketwatch.com/invest	0	NYSE	1178.	1174.	1191.	1174.	1177.86	0.03	161142	684.	1274.	16.3	27.54323	0.9245611	
AUTO	HMC	Honda Motor Co Lt	CONSHLTHP	https://www.marketwatch.com/invest	0	NYSE	24.4	24.4	24.4	24.3	24.12	1.24	430007	19.3	29.4	22.5	4760.43	0.829483	
CANNAB	ACB	Aurora Cannabis	CONSHLTHP	https://www.marketwatch.com/invest	0	NYSE	4.62	4.6	4.81	4.53	4.59	0.65	6974089	4.5	55.6	#N/	0.746188	0.0829741	
CANNAB	CGC	Canopy Growth	CONSHLTHP	https://www.marketwatch.com/invest	0	NYSE	14.9	14.1	15.0	14.1	14.05	6.33	4281037	9	25.97	#N/	7.358610	0.5752791	
CANNAB	SMG	Scotts Miracle-Gro	CONSHLTHP	https://www.marketwatch.com/invest	0	NYSE	153.	151.	154.	151.	150.75	1.54	243235	76.5	176.0	26.7	8.53653	0.869320	
CANNAB	CRON	Cronos	CONSHLTHP	https://www.marketwatch.com/invest	0	NASDA	5.24	5.02	5.25	5.02	5.03	4.17	2941007	4	10.56	#N/	2.42820	0.4962121	
CANNAB	CURA	Curaleaf	CONSHLTHP	https://www.marketwatch.com/invest	0	CNSX	9.53	9.41	9.56	9.41	9.51	0.21	349616	3.72	12.75	#N/	4.197158	0.747450	
CONG	DIS	Disney	CONSHLTHP	https://www.marketwatch.com/invest	0	NYSE	123.	123.	123.	122.	122.55	0.67	5919150	79.0	153.4	#N/	222.9373	0.804184	

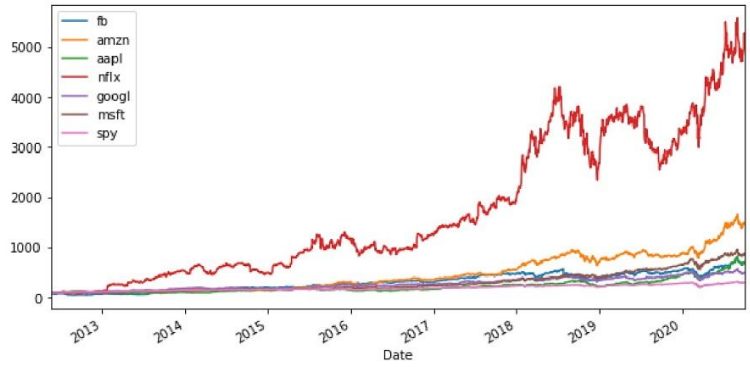
Figure 5.3.3: BASIS's Setter interface showing the Question Library



5.4 Blackbox

For the interface between the classes Portfolio and Allocator, Jupyter Notebook is chosen. Since the client does not interact with the blackbox (backend), the interaction is just for testing purposes. An example of testing of certain functions is in the snapshot below.

```
In [107]:
import ffn # function for quantitative analysis in python
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
from empyrical import alpha_beta

In [108]:
#calculate mean variance for faang stocks + microsoft and S&P500
# to get price at open do x:Open where x is ticker. Similar for close. Note:
price = ffn.get('fb, amzn, aapl, nflx, googl, msft, spy').to_returns().dropna

mean_var = price.calc_mean_var_weights().as_format('.2%')
mean_var
#msft = ffn.get('MSFT', start = '2010-01-01')
#spy = ffn.get('SPY', start = '2010-01-01')

Out[108]:
fb        0.00%
amzn     44.00%
aapl     22.05%
nflx      5.43%
googl     0.00%
msft     28.53%
spy       0.00%
dtype: object

In [109]:
price = ffn.get('fb, amzn, aapl, nflx, googl, msft, spy')

In [110]:
#comparing performance of stocks relative to each other
rel = ffn.get('fb, amzn, aapl, nflx, googl, msft, spy', start = '2010-01-01'
type(rel)

Out[110]:
matplotlib.axes._subplots.AxesSubplot
```



6. SelectorFetcher

The purpose of the SelectorFetcher Interface is to enable seamless integration between portfolio allocation and the selection of stocks. SelectorFetcher shall be implemented by SSeletor and Fetcher and be accessed by Portfolio class.

Appendix A: Inspection Report

Date	Comments	Status	Inspector
10/05/2020	Attributes and Methods for 3. Principal Classes to be added	Completed Vinayak Mathur	Anurag Upadhya
10/06/2020	Revise 5. Class Interfaces	Completed Vinayak Mathur	Anurag Upadhya
10/08/2020	Diagrams to be added to 4. Inter-Agent Protocols	Completed Vinayak Mathur	Anurag Upadhya