Stock Portfolio Risk Analyzer

Table of Contents

- Description
- Process
- Requirements & Specifications
  - User Stories: Iteration 2
  - User Stories: Iteration 3
  - User Stories: Iteration 4
  - User Stories: Iteration 5
  - User Stories: Iteration 6
  - Use Cases
- Architecture & Design
- Reflections and Lessons Learned
- Important Links

TEAM SPRA:

Rohan Kapoor (rkapoor6)
Laurynas Tamulevicius (tamulev2)
Alex Ashley (ahashle2)
Enrique Espaillat (espaill2)
Thibaut Xiong (txiong3)
Shivam Gupta (sgupta46)
Arpit Dev Mathur (admathu2)
Ronit Chakraborty (rchakra2)

Assigned T.A.: Yiming Jiang

Description

An increasing number of investors are managing their investment portfolios by themselves these days and thus have less idea on how risky their portfolio is. This is an issue because they don't have access to the advanced tools investment companies use, thus they cannot infer knowledge from the historical data. Our service analyses a given portfolio and computes how risky other stocks are relative to their current portfolio. This will greatly influence investors' decision based on his risk appetite by giving him deeper insight in the market.

Being investors ourselves, we usually end up making sentimental buy and sell decisions without actually analysing the consequences it may have on our portfolios. This project is interesting because it may help investors make much more informed investment decisions based on statistics and not just sentiments. We are passionate about this project because we see the value of what it may bring to our investment lives.

Process

We closely followed the XP software development process. Clean, working code was our primary deliverable and we wrote both back-end and GUI tests to make sure that the code did what it promised. Every week, we split up either in pairs or triples to work on 2-3 user stories and we rotated every week.

There were weekly group-wide meetings where we updated each other on the progress of our user stories and at the end of iterations, assigned new user stories to pairs. These meetings also served as a forum to discuss any issues that we faced in working on code so the productivity of the group was optimal.

For communication we used Slack and the Project wiki. Day to day communications would take place on Slack. The biggest advantage of Slack is the ability to form individual channels for communication within the pairs and then we had one general channel for the entire team. The project wiki was useful for keeping up to date with the progress of

other pairs. Both of them were used in conjunction to make sure everyone was on the same page.

For version control and collaboration, we used git (and Github for hosting). We chose Github because most of us were familiar with it and preferred it over other options like SVN. Github also includes graphical ways of doing steps of the git process such as branching, merging, and pull requests. Integration with Travis CI to test our build at each commit and pull-request was also very helpful in ensuring that our build was stable and consistent every time new features/code was added.

We also adhered to one of the other core principles of XP which was arriving at the simplest solution before adding extra functionality. We took it one iteration at a time, focusing only on accomplishing the requirements of the current user story. As we moved forward, we made the required adjustments in response to a changes in the requirements.

Requirements & Specifications

User Stories

Iteration 2

actual	estimated	story description	pair
3 units	3 units	Get ourselves familiar with Yahoo Finance API	Shivam/Thibaut
3 units	3 units	Get ourselves familiar with Quandl	Larry/Alex
4 units	4 units	get_data methods (Yahoo Finance)	Shivam/Thibaut
4 units	4 units	get_data methods (Quandl)	Larry/Alex

3 units	3 units	Front-end design brainstorming session	Rohan/Arpit
3 units	3 units	Back-end design decision	Larry/Alex
3 units	4 units	Front-end dashboard mockup	Ronit/Enrique

Iteration 3

actual	estimated	story description	pair
7 units	8 units	Obtain Relative Risk Index (RRI) based on a portfolio ● Do research ● Calculate relevant risks	Larry and Shivam
4 units	4 units	Setup Django w/ Heroku	Rohan and Thibaut
4 units	4 units	Front and back end integration	Ronit and Alex
2 units	3 units	Interactive front end widgets	Ronit and Alex
4 units	4 units	User registration	Enrique and Arpit

Iteration 4

actual	estimated	story description	pair
5 units	4 units	Modify User account	Larry, Alex and Arpit
4 units	4 units	Allow user to modify portfolio	Ronit, Enrique, Thibaut
5 units	5 units	Create Back-End DB Models and REST API	Ronit, Enrique, Thibaut

7 units	6 units	Make dashboard UI dynamic: ● Plotting the risk history ● Donuts chart ● Gravatar integration	Larry, Alex and Arpit
6 units	6 units	Work on RRI algorithm	Shivam and Rohan
2 units	2 units	Make a landing page	Shivam and Rohan

Iteration 5

actual	estimated	story description	pair
6 units	5 units	multiple portfolios	Rohan, Enrique
6 units	6 units	stock recommendation	Alex, Thibaut
4 units	4 units	stock interface	Larry, Shivam
4 units	4 units	search bar	Arpit, Ronit
3 units	3 units	ranking	Rohan, Enrique
2 units	2 units	change password	Arpit, Ronit
2 units	3 units	newsfeed	Larry, Shivam

Iteration 6

actual	estimated	story description	pairs
2 units	2 units	Integrate Quandl API	Thibaut, Shivam
3 units	4 units	Improve stock search and connect search results with stock interface	Larry, Rohan, Ronit

3 units	2 units	Extend stock interface	Thibaut, Shivam
3 units	3 units	Upload/download portfolio w/ RRI(csv)	Alex, Enrique, Arpit
3 units	3 units	Email newsletter	Alex, Enrique, Arpit
4 units	2 units	Cache stock graph data	Larry, Rohan, Ronit
4 units	4 units	Portfolio simulation	Thibaut, Shivam
4 units	4 units	Top 10 portfolio ranking view	Larry, Rohan, Ronit
4 units	3 units	Portfolio generation	Alex, Enrique, Arpit

Use Cases

Use case	Modify account
Primary Actor	User (Investor)
Goal in Context	User can change account information such as password and email
Precondition	Verified user account
Minimal Guarantees	A user will be able to change his email address and password
Triggers	1. Investor clicks on modify account. 2. Investor can now change password or email by clicking on those.
Extensions	1. If this password is too short, the prompt will be raised. 2. The user will have to enter a valid email to switch to

Use case	Create an account
Primary Actor	User (Investor)
Goal in Context	User can access the website sign up to create an account.
Precondition	Active email address
Minimal	No one will have same username and the verification email will be sent when

Guarantees	registered
Triggers	1. Investor clicks on Sign-up 2. User enters username, email and password 3. User clicks Submit 4. User clicks a link sent to his email
Extensions	1. If the password is too short the prompt will be raised 2. If username is not unique it will ask to add another one.

Use case	View relative risk of portfolio
Primary Actor	User (Investor)
Goal in Context	Compare portfolio risk to other investors.
Precondition	User has a registered account and at least one portfolio
Minimal Guarantees	The user will be able to see their riskiness.
Triggers	The user clicks the top portfolios button
Extensions	None

Use case	Generate Portfolio
Primary Actor	User (Investor)
Goal in Context	User can generate a new portfolio, based on their current default portfolio. The value of the generated portfolio is ideally with a 40% range of the current default portfolio.
Precondition	User has a registered account and is on the dashboard.
Minimal Guarantees	A portfolio will be generated for the user which the user can save as a new portfolio.
Triggers	The user attempts to create a new profile and clicks the “generate portfolio” button and selects if they want it or not.
Extensions	None

Use case	Portfolio Rank
Primary Actor	User (Investor)

Goal in Context	User can see his Portfolio Rank based on the Risk Value of his portfolio compared to all of our app users.
Precondition	User has a registered account and is on the dashboard.
Minimal Guarantees	The user will be able to see the rank of the selected portfolio.
Triggers	The user's loads the portfolio by selecting from the side-menu of the dashboard.
Extensions	None

Use case	View portfolio info in a graphical format
Primary Actor	User (Investor)
Goal in Context	User can choose to display portfolio information in a graph: Graph of Risk History (Historical), Pie Chart (Sector Diversity, Portfolio Diversity)
Precondition	Must be registered Must have at least one portfolio
Minimal Guarantees	Will always display a graph
Triggers	The dashboard is loaded
Extensions	The data can be replotted by selecting time range

Use case	Upload portfolio
Primary Actor	User (Investor)
Goal in Context	User can access the website and upload a valid csv file which will in turn generate a portfolio
Precondition	Must be registered user Must possess a valid csv file
Minimal Guarantees	The portfolio is not created if incorrect csv file is uploaded
Triggers	Press Import Portfolio button
Extensions	None

Use case	Download portfolio
Primary Actor	User (Investor)
Goal in Context	User can access the website and download a csv representation of his portfolio. (For backup purposes.
Precondition	Must be registered user Must posses a valid csv file
Minimal Guarantees	A valid csv file should be downloaded. This same downloaded csv file should be valid for using within the upload portfolio feature.
Triggers	Press Download Portfolio button
Extensions	None

Use case	Simulate Portfolio and compare against Market.
Primary Actor	User (Investor)
Goal in Context	The user generates a simulated portfolio and compares the new portfolio to general market trend (risk, price, and money generated). Information is displayed in graphs.
Precondition	User is registered and has at least one portfolio to simulate results for.
Minimal Guarantees	Will display graphs of: 1. Portfolio returns vs S&P 500 returns 2. Beta-weighted portfolio returns vs S&P 500 returns 3. Monthly aggregated heatmap of portfolio returns 4. Yearly aggregated bar-chart of portfolio returns 5. Monthly distribution of portfolio returns
Triggers	The user clicks on the ‘Simulate Portfolio’ button.
Extensions	The user can repeat the same process for any of their saved portfolios.

Use case	Email newsletter
Primary Actor	User (Investor)

Goal in Context	Send information to all users through a weekly email newsletter. It will include risk, price, and rank value
Precondition	User is registered and has at least one portfolio save.
Minimal Guarantees	Will send an email to user's registered email with the following information: 1. Portfolio risk 2. Portfolio value/prices of holdings 3. Rank
Triggers	The user clicks on the 'Simulate Portfolio' button.
Extensions	The user can repeat the same process for any of their saved portfolios.

Use case	Stock Lookup
Primary Actor	User (Investor)
Goal in Context	User can search for a stock and compare it to the overall risk factor of the user's portfolio. Also, display information on the stock (RSS feed, risk graph, price graph, sentiment).
Precondition	User is registered, and logged in. Also, has a portfolio selected in his dashboard.
Minimal Guarantees	1. Autocomplete will show up as soon as you start typing.
Triggers	The user types on the search bar. (Autocomplete shows up) Hitting enter on the keyboard, or the search UI button launches a modal.
Extensions	If stock doesn't exist the Modal will not show to the user. The company can be looked up by its name or ticker.

Use case	Top Portfolios
Primary Actor	User (Investor)
Goal in Context	Users can see the public information (RRI, Value, and Name) of the top 8 portfolios for the following categories: "Most Risky", "Less Risky", "Most Valuable", "Least Valuable". If user clicks on a portfolio, he can see the list of stocks and the corresponding information of the specified portfolio (Company, Symbol, Sector, Quantity, Last Price, Market Value)

Precondition	User is registered, and logged in. They must be on the dashboard page.
Guarantees	The top 10 portfolios for the following categories will be displayed on the dashboard: 1. “Most Risky” 2. “Less Risky” 3. “Most Valuable” 4. “Least Valuable” The user should also be able to get the full portfolio information after clicking on one of the portfolios.
Triggers	The user loads the dashboard.
Extensions	None

Use case	Multiple Portfolios
Primary Actor	User (Investor)
Goal in Context	Allow a user to create more than one portfolio, he can then proceed to select a portfolio to view all the information associated with it and access all of the app’s features oriented towards a specific portfolio.
Precondition	The user must be registered and logged in.
Minimal Guarantees	The modal will show always show up and user can always have more than 1 portfolio.
Triggers	The user creates more than 1 portfolio.
Extensions	The user can create unlimited portfolios.

Use case	Modify portfolio information
Primary Actor	User (Investor)
Goal in Context	User can modify his portfolio, stocks and quantities.
Precondition	Must have portfolio
Minimal Guarantees	The modal will show always show up
Triggers	The modify portfolio button is pressed
Extensions	N/A

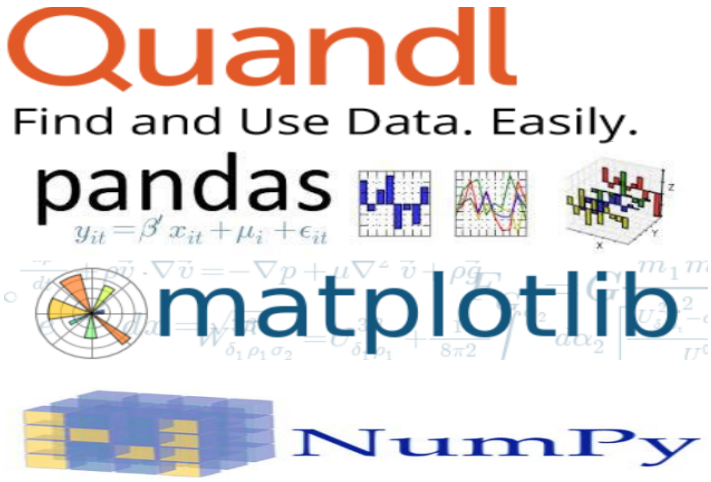
Use case	Stock Recommendations
Primary Actor	User (Investor)
Goal in Context	User can get stock recommendations to add to his currently selected portfolio for the following categories: Stability (Risk Neutral Stocks), Variety (Diversify his currently selected portfolio), Relax (Lower Risk of current portfolio), Gamble (Increase Risk of currently selected portfolio).
Precondition	Must have portfolio
Minimal Guarantees	Will always provide recommendations
Triggers	Press Stability , Variety, Relax, Gamble buttons
Extensions	Can add selected stock to portfolio

Architecture & Design

The project is structured like a typical Django project with the html templates being grouped together in the templates folder. Django's built-in url router works from the urls.py file which contains all the information needed for routing. To serve to our clients all of our back-end functionality we decided to build a thorough REST API, which is routed by our `urls.py` under the api module and served by our api.py file. The REST API is secure so that only authenticated users are able to make requests, if and only if they have permission to access the assets (i.e. get portfolio information, add stock to portfolio, etc.)

Our application follows the standard Django MVC architecture. The url router as implemented in urls.py in conjunction with the `views.py` represent the controller in the MVC architecture. The Model is represented by the backend database models which in turn, represents our database schema. The views are organized under templates and the code is designed to be reusable. By building on a common base template, we were able to reuse html elements that are common to a lot of different pages.

For our back-end functions we used libraries such as Yahoo Finance (www.finance.yahoo.com/), Quandl (www.quandl.com), NumPy, Pandas, matplotlib, and seaborn to power our time-series calculations as well as visualization. These tools are standard in building financial applications and provide many built-in methods for common financial calculations.



Our market data (as well as additional information about stocks) came from the Yahoo Finance and Quandl. Fortunately, both of these sources provided free intraday stock data (more granular data is usually expensive to purchase). In most cases we loaded time-series market data from these APIs into Pandas DataFrames and Series which provide vectorized (numpy arrays) arrays which are useful for many types of time-series calculations we needed to do. Most of these functions are located in the datautils directory. In addition to data pulled from APIs, we also included some static files that we gather information from such as `secwiki_tickers.csv` which is a composite list of all stocks we handle and also contains information about those companies. In general, the api methods call these datautil methods to pull the necessary data, then push a response to a page requested by the model. Data-processing that powered the app on the backend include:

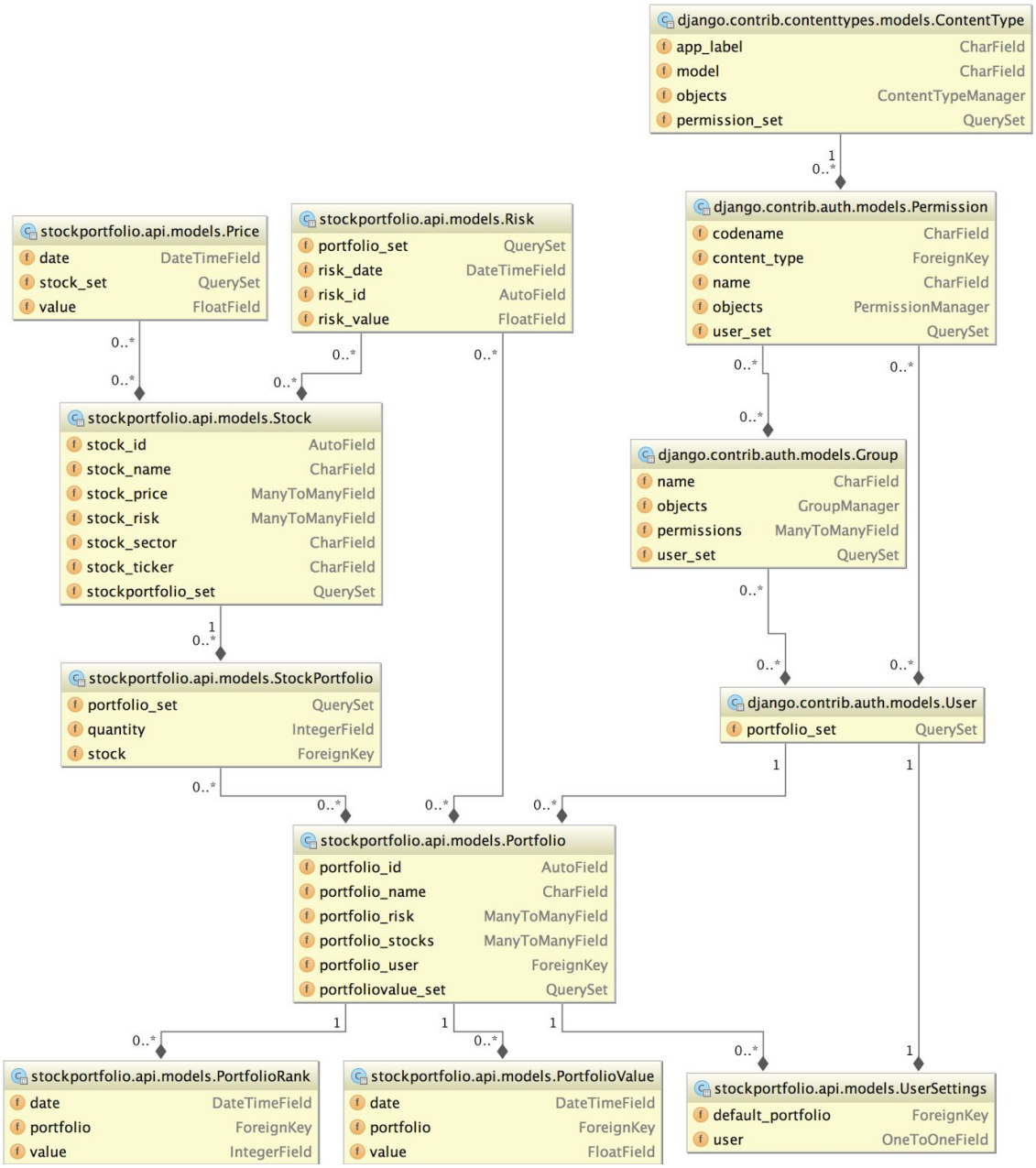
1. `quandl_info.py` (pulling stock data from Quandl API)
2. `sentiment.py` (pulling sentiment information from Quandl API)
3. `yahoo_finance.py` (pulling stock data from Yahoo Finance API)
4. `stock_info.py` (uses numpy/pandas to perform risk calculations)
5. `portfolio_simulation.py` (uses pandas/matplotlib/seaborn to simulate+visualize portfolio)

Our REST back-end functions are all inside:

1. `api.py` (Functions)
2. `api/urls.py` (Routing for api functions)

In order to provide users with historical risk, rank and price values for their stocks and portfolios in the most efficient way possible we have CRON Jobs in our Heroku Server Instance. In other words, all of the stock market data and risk values are being stored after computations are completed by our server on a daily basis. Our team decided to do this in order to avoid doing external API calls or risk/rank calculations on the fly. This obviously increased our demand for database space, but we believe this was the best course of action to provide our users with a better experience.

SPR Web App Database Models:



Django Models Run-Down:

`Risk` - Holds Risk information for a specific portfolio or stock, and when it was computed.
`Price` - Holds Price information for a specific Stock, and when it was calculated.
`Stock` - Holds all Stock Information for any given stock (name, all the price objects associated with the Stock, all the Risk objects associated with the Stock, sector, ticker)
`StockPortfolio` - Holds a Stock Object and a quantity for that stock.

`Portfolio` - Holds a User's Portfolio Information. (Name, List of Risk Objects calculated for portfolio, list of StockPortfolio Objects for that stock, User Object associated with the portfolio)
`UserSettings` - Holds a foreign key to the default portfolio of a given user, and the User Object.
`PortfolioRank` - Holds an integer value representing the rank of a portfolio and the date when it was computed.
`PortfolioValue` - Holds an float value representing the value of a portfolio and the date when it was computed.

Reflections and Lessons Learned

Rohan Kapoor (rkapoor6): Version control and communication are two very large things. Most of our team didn't have much experience with either of them. I ended up taking on the role of project manager and build engineer making sure that people were breaking apart the user stories correctly and enforcing code standards on them.

Laurynas Tamulevicius (tamulev2): Team taught me to appreciate active people, who are responsible enough to attend meetings, contribute to development and come to meetings on time. I was introduced to many new tools and learned to manage the project. I learned to reasonably split the workload among active team members when some teammates “disappeared”.

Alex Ashley (ahashle2): Working on the SPRA team reinforced the importance of team communication. While we got off to a rough start, once we started communicating and meeting regularly, the process became much smoother.

Enrique Espaillat (espaill2): Working in a large team setting demanded that we mastered our Version Control System (Github) in order to avoid breaking each other’s code. We used feature branches to implement each of our own user stories, and met constantly to correctly merge all of the conflicts generated. Working on this project also reinforced the need to properly comment all your code, given that other people often need to understand it as soon as you push your code into production.

Shivam Gupta (sgupta40) While working on the SPRA team I learned to use Github quite well, given I had very little experience beforehand. I also learned to manage my time accordingly and to communicate well with other team members of the team on my progress. This allowed me to avoid code conflicts with other pairs of the team.

Thibaut Xiong (txiong3): Commenting and writing clean code was very important in working in a large group setting. Often, we would have to go into code that other people wrote and use their methods/design choices. It was important that they made it clear and understandable so that we could instantly comprehend what was going on.

Ronit Chakraborty (rchakra2): Maintaining clear lines of communication between team mates and writing maintainable code is very important. Commenting code well is essential for easy code maintenance. Starting well ahead of time is important in order to avoid running into issues near the deadline.

Arpit Dev Mathur (admathu2): Communication and version control are the two big takeaways. Regular and honest communications are required to ensure smooth running of the group. Working with various branches and merging them every week needed a certain level of expertise and communication.

Important Links

- Github Repo
  - Coveralls Page (Code Coverage)
  - Travis CI Page (Continuous Integration)
  - Code Documentation
- Wiki Page
- Notes Page