

---

# **Final Report**

**for**

## **Capstone: S&P 500 Earnings Price Data Scraper, Calculator, and Calendar**

**Prepared by Ann Katz**

**COP-2939-48329**

**Dr. Banisakher**

**March 24, 2021**

# Table of Contents

<b>Table of Contents.....</b>	<b>ii</b>
<b>1. Overview.....</b>	<b>1</b>
1.1 Purpose of Document.....	1
1.2 Overview of Report.....	1
1.3 Original Purpose of Program.....	1
<b>2. Program at Runtime.....</b>	<b>2</b>
2.1 Overview of Program Package.....	2
2.2 Overview of Installation and Execution.....	2
2.3 Walkthrough and Sample Photos.....	3
<b>3. Requirements Analysis.....</b>	<b>5</b>
3.1 Compliance to Requirements Specification and Test Matrix.....	5
3.2 Other Testing.....	9
3.3 Obstacles to Meeting Requirements & Errors Encountered During Tests.....	9
3.4 New Requirements/Added Features.....	10
3.5 Potential Future Requirements/Features.....	10
<b>4. Design of Program.....</b>	<b>11</b>
4.1 Design Overview.....	11
4.2 Comparison with Original Design Document.....	13
4.3 User Interface Differences.....	14
4.4 Design Obstacles.....	16
4.5 Python Performance and Packages Utilized.....	17
<b>5. Summary of Program and Findings.....</b>	<b>17</b>
5.1 Summary of Capstone Project Creation.....	17
5.2 What Went Right.....	17
5.3 Lessons Learned.....	18
5.4 What's Next For The Project.....	18
<b>6. Appendix: Resources.....</b>	<b>18</b>

# **1. Overview**

## **1.1 Purpose of Document**

The purpose of this document is to report, in detail, the Capstone Project's creation process, design, execution, and usability. It compares the final project to the previously defined requirements and use cases, along with the early design document. Its purpose is to explain why certain decisions were made when it came to design, where the program could be improved, and what went according to plan. It also goes into detail about new features or tasks added, along with obstacles confronted throughout the process. A walkthrough demonstration video and photos are also provided to show the program at runtime.

## **1.2 Overview of Report**

The document begins, in this Section (1) with an overview of both the Final Report (this) document and the program's purpose.

Section 2 describes the program at runtime, the installation process or requirements necessary to install the executable program file. It overviews the source code files as well. It also shows photos of the program's opening and user interface, as well is a demonstrational video in which I perform every test in the Test Matrix.

Section 3 goes over the performance of the program in relation to the original requirements. It also describes other real-world tests performed to determine accuracy. It reviews the obstacles faced when it came to meeting requirements. It also goes over other features I added and other features I plan on adding in the future.

Section 4 goes into detail about the final design choices made in the program, and how the source code differs from the original design plan. It also recounts the barriers faced when it came to source code, UI design, and the performance of the Python language and the libraries used.

Section 5 summarizes the project and final program. It recounts the successes of the project, lessons learned throughout creation, and the future of the program.

## **1.3 Original Purpose of Program**

The program's original purpose in the proposal was to provide a simple Windows executable for options traders and anyone else interested in United States stock market performance related to S&P 500 companies. Personally, I needed a program that could easily show me upcoming earnings dates for certain companies and their past performances post-earnings reports, as I use that data to buy and sell stock options. It had previously taken me hours to perform the calculations and research myself, and I could not find a program that would do it for me and save me time. Please note that the program was intended for my use, and although anyone is welcome to use it, the program, as well as any related report, is not intended to provide investment advice or stock price predictions or estimations of any kind.

According to the Requirements document, the program was intended to be a Windows desktop application (created with the Python 3 language) that, when executed, was to provide a interface that would allow users to search through/enter their choice of current S&P 500 companies and fetch (the past ten) historical earning report dates, times, and prices for those 500 companies. It was to use that data to calculate the average 24-hour price (U.S. Dollar) difference from the market day before the past ten earnings report to the corresponding market day after, and the percentage difference. The app was also to scrape and list the dates of the next upcoming earnings reports. It was intended to be easy to use and install.

## **2. Program at Runtime**

### **2.1 Overview of Program Package**

The program package, named “sp-500-earnings” contains multiple files, including Python files, a pickle file for holding data at runtime, a requirements document for easy installation, a folder of icons for the GUI, and a README text file with installation instructions, command line instructions, API documentation, and program features.

### **2.2 Overview of Installation and Execution**

Included in the program README text are installation instructions to create an executable Windows file, instructions on running the program from the command line, and other ways one may use or view the program source code. A Python 3 distribution and PATH environment variable is necessary in nearly every scenario.

Please note: The program file to run (in most scenarios other than executable installation) is the gui.py file. In almost every scenario, the program will take a few moments to run. It is faster in some virtual environments. Within the source code, if running the gui.py file, I included a console progress bar of program open.

To install the executable, the instructions are as follows: Make sure you have downloaded Pyinstaller (via Pip or download), an updated pip PATH environment variable, and Python 3. Then Navigate to program folder “sp-500-earnings” in the command line. Run "python install.py" (or "python3 install.py"). This will install all requirements from the requirements document and create a folder within the program called “dist” that holds the executable along with the pickle and icons files necessary. Double click the executable to run the user interface.

To run from the command line, the instructions are as follows: When running the program, the user needs to install the requirements first. This is easily done by use of the requirements.txt file, which can install every Python package necessary with Pip. After acquiring/upgrading pip and adding it to the PATH, install all the necessary dependencies by running (from the sp-500-earnings folder path): pip install -r requirements.txt. Then, to run the user interface, navigate to the sp-500-earnings folder path and run the gui.py file in the command line with the command: python gui.py (or python3 gui.py). Or you may use a virtual environment with the packages necessary.

The README file also provides the API documentation and at-runtime instructions. The at-runtime instructions are defined within the program walkthrough.

## 2.3 Walkthrough and Sample Photos

Instructions for using features are as follows as defined in the README file:

- Sort columns by clicking the 'SORT-BY' buttons on the right side of the page
- Columns can also be clicked to be sorted.
- Search for a company with the search bar and search button.
- Navigate to home, help, or exit the program from the right side of the window.
- The current price for each company is updated upon program open.
- Choose a company to display:
  - Earnings data for the past 10 earnings releases including
  - The average stock price change (in dollars and percent) for the 24-hours in market day after earnings report
  - Exact price changes for the past 10 earnings by dollar amount and percentage
  - An interactive stock chart marking the past earnings. Blue triangles represent the change at earnings dates.

Further explanations of functionality can be seen in the Use Cases table in Section 3.1

Photos of Program at Runtime:

Home Page:

Symbol	Company Name	Current Price	Average Change (USD)	Percent Average Change	Upcoming Earnings Date
A	Agilent Technologies	136.68	1.0	1.28	2021-05-20
AAL	American Airlines Group	21.11	0.51	2.41	2021-04-29
AAP	Advance Auto Parts	199.75	2.33	1.64	2021-05-18
AAPL	Apple Inc.	134.32	0.29	1.13	2021-04-28
ABBV	AbbVie Inc.	111.38	1.05	1.26	2021-04-30
ABC	AmerisourceBergen	121.65	1.55	1.85	2021-05-06
ABMD	Abiomed	351.03	1.04	1.03	2021-04-29
ABT	Abbott Laboratories	123.31	0.06	0.0	2050-01-01
ACN	Accenture	291.74	3.06	1.53	2021-06-24
ADBE	Adobe Inc.	515.84	2.49	1.11	2021-06-10
ADI	Analog Devices, Inc.	159.02	2.02	2.02	2021-05-19
ADM	Archer-Daniels-Midland (	59.41	-0.57	-1.21	2021-05-05
ADP	Automatic Data Process	195.86	0.42	0.48	2021-04-28
ADSK	Autodesk Inc.	295.27	1.29	0.9	2021-05-26
AEE	Ameren Corp	84.78	-0.44	-0.65	2021-05-10
AEP	American Electric Power	87.7	-0.36	-0.49	2050-01-01
AES	AES Corp	28.3	-0.26	-1.0	2021-05-06
AFL	Aflac	53.37	-0.11	-0.44	2021-04-28

## Help Screen:

**IN PROGRAM FEATURES:**

- Sort columns by clicking the 'SORT-BY' buttons on the right side of the page. Columns can also be clicked to be sorted.
- Search for a company with the search bar and search button.
- Navigate to home, help, or exit the program from the right side of the window.
- The current price for each company is updated upon program open.
- Choose a company to display:
  - Earnings data for the past 10 earnings releases including
  - The average stock price change (in dollars and percent) for the 24-hours in market day after earnings report
  - Exact price changes for the past 10 earnings by dollar amount and percentage
  - An interactive stock chart marking the past earnings. Blue triangles represent the change at earnings dates.
- API Documentation:
  - \*Please note: for some URLs shown below, the example ticker of TSLA is used.

Symbol	Company Name	Current Price	Average Change (USD)	Percent Average Change	Upcoming Earnings Date
A	Agilent Technologies	136.68	1.0	1.28	2021-05-20
AAL	American Airlines Group	21.11	0.51	2.41	2021-04-29
AAP	American Express	155.00	0.00	0.00	2021-05-18
AAPL	Apple Inc.	130.00	0.00	0.00	2021-04-28
ABBV	Abbott Laboratories	130.00	0.00	0.00	2021-04-30
ABC	Amgen Inc.	130.00	0.00	0.00	2021-05-06
ABMD	Abbot Medical Devices	130.00	0.00	0.00	2021-04-29
ABT	Abbott Laboratories	130.00	0.00	0.00	2021-05-10
ACN	Accenture	130.00	0.00	0.00	2021-06-24
ADBE	Adobe Inc.	130.00	0.00	0.00	2021-06-10
ADI	Analog Devices	130.00	0.00	0.00	2021-05-19
ADM	Archer-Daniels-Midland	130.00	0.00	0.00	2021-05-05
ADP	Automatic Data Processing	130.00	0.00	0.00	2021-04-28
ADSK	Autodesk	130.00	0.00	0.00	2021-05-26
AEE	Ameren Corp	84.78	-0.44	-0.65	2021-05-10
AEP	American Electric Power	87.7	-0.36	-0.49	2050-01-01
AES	AES Corp	28.3	-0.26	-1.0	2021-05-06

## Search Result after typing in a portion of ticker:

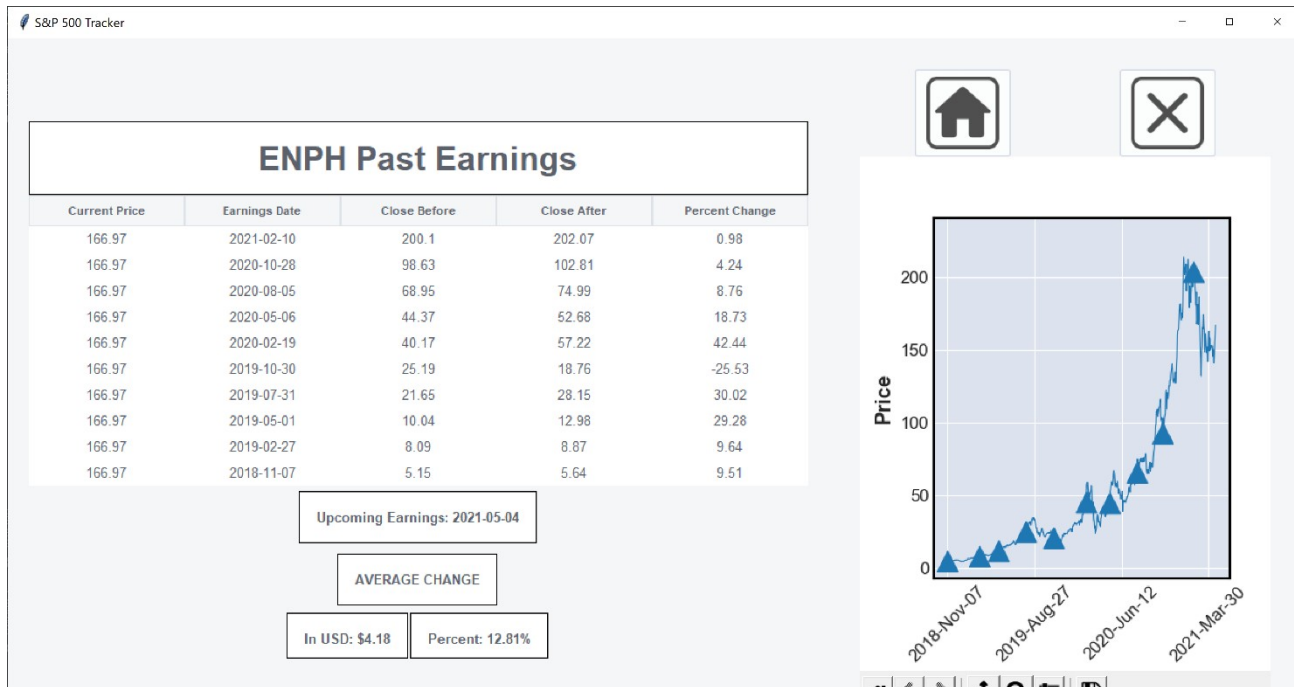
**Current S&P 500 Companies**

Symbol	Company Name	Current Price	Average Change (USD)	Percent Average Change	Upcoming Earnings Date
A	Agilent Technologies	136.68	1.0	1.28	2021-05-20
AAL	American Airlines Group	21.11	0.51	2.41	2021-04-29

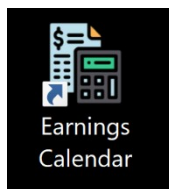
Search results for 'en':

Symbol	Company Name	Current Price	Average Change (USD)	Percent Average Change	Upcoming Earnings Date
ENPH	Enphase Energy	166.97	4.18	12.81	2021-05-04
ETR	Entergy Corp.	106.71	0.02	0.1	2021-05-10

Company Information Page after clicking a search result:



In an April 26, 2021 update, I provided for the installation file `install.py` to also create an icon (free to use from [freepik.com](https://www.freepik.com)) for the executable file. The program shortcut can be renamed as well.



### 3. Requirements Analysis

#### 3.1 Compliance to Requirements Specification and Test Matrix

Below is a chart of the Testing Matrix, including the Use Cases and testing method. If passed, a check mark will appear in the fourth column. If failed, an "X" will appear. If ambiguous, a tilde "~" will be present and the ambiguity will be explained below, along with either a plan to make the program pass the requirement or a change to the requirement. In retrospect, the tests should have been made smaller to avoid ambiguity.

## Capstone: S&amp;P 500 Historical Data Fetcher, Calculator, and Calendar

Ann Katz

<i>ID</i>	<i>Requirement</i>	<i>How to Validate</i>	<i>PASS</i>	<i>FAIL</i>
UC1	User opens program: User clicks the icon/program from either their files, start screen, or desktop (wherever they have a clickable application file saved). Window opens and home screen appears.	I will click on the desktop icon shortcut to the program to show that the program and user interface opens functionally and without error. I will then repeat this process using a variety of opening methods. This includes opening the program from the start screen, ensuring it opens without error, then closing it. Then I will open it from the command line to ensure it opens and runs without error and then close it. Then, I will open the program from my Files to ensure it opens and runs without error.		
UC2	User searches for a company: User clicks into the search box and types either the company name or company ticker and presses search. A new window appears with a list of matching companies, from which the user may click. If a company ticker is entered that does not match any current company within the S&P 500 index, an error message appears. The user may click the Home button to try again.	With the program open, I will click into the search box and show that I may enter text to search. I will enter a company name and then press the search button. I will show that the company appears in the list of results. Then, I will go back to the search bar, this time with the ticker of another company. I will press the search button and show that the company appears in the list of results. I will then go back to the search bar and type in a portion of a company name, then press the search button to show that the company appears in the list of search results. I will go back to the search bar and enter the name of the company that is not within the index. I will press the search button to show that an error message appears, and that no company appears in the list. I will close the error message.		



UC3	User chooses company: After the user clicks on a company ticker or company name from either the list on the home screen or after using the search button, the clicked company's information page appears.	From the home screen, I will click on a company ticker from the list to show that the company's information page appears. I will then go back to the home screen and click on another company's name from the list to show that their information page appears. I will go back to the home page and search for a company using the search bar. From the results, I will click on the company's name and ticker to show that the company's information page appears.		
UC4	User clicks the "Sort by... Upcoming Earnings Date" button: From the home screen, the user clicks the button to sort the entire list in order of upcoming earnings date, from nearest to farthest. The list remains clickable and if any company is clicked, it will take the user to the company's information page.	From the home screen, I will click the "Sort by...Upcoming Earnings Date" button to show that the list on the side updates. From there, I will click the first company to be taken to their information page to note the next earning's date. I will repeat this with the next four companies in the sorted list. It should show that the companies' next upcoming earnings dates appear in order of nearest to farthest.		
UC5	User clicks the "Sort by... Current Price" button: From the home screen, the user clicks the button to sort the entire list in order of current price per stock, from highest to lowest. The list remains clickable and if any company is clicked, it will take the user to that company's information page.	From the home screen, I will click the "Sort by...Current Price" button to show that the list on the side updates. From there, I will click the first company to be taken to their information page to note their current price. I will repeat this with the next four companies in the sorted list. It should show that the companies' prices appear in descending order by most expensive stock to least expensive stock.		
UC6	User clicks the "Sort by... Average % Change Post-Earnings" button: From the home screen, the user clicks the button to sort the entire list in order of highest percentage change in stock price 24 hours after earnings, from highest to lowest. The list remains clickable and if any company is clicked, it will take the user to that company's information page.	From the home screen, I will click the "Sort by...Average % Change Post-Earnings" button to show that the list on the side updates. From there, I will click the first company to be taken to their information page to note their average percent change. I will repeat this with the next four companies in the sorted list. It should show that the companies' average percent changes appear in order of highest change to lowest, percentage-wise.		

UC7	User clicks the "Home" button: In the top right corner of every page is a home button, to take the user back to the home page at any time	I will navigate to a company information page. I will click the home button to see that I am taken back to the original home page (with the list in alphabetical order by ticker). I will sort a list using one of the sorting buttons and then click the home page. This will show that the list on the side updates back to alphabetical order and that none of the sorting buttons are clicked. I will navigate to the "Help" screen and then click the home button. This will take me back to the original home screen.		~
UC8	User clicks the "X" button: In the top right corner of every screen is a typical "X" exit button. When clicked, the program terminates and the window is closed.	From the home screen, I will click the "X" button to show that the program closes completely and stops running. Then I will reopen the program and repeat the process from another screen to show the same result.		
UC9	User clicks the "Help" button: The "Help" button is present in every screen within the program, located in the bottom right corner. When clicked, it takes the user to the Help screen.	From the home screen, I will click the "Help" button to show that it takes me to a Help screen, with a list of information about the program, along with what each button in the program does, what each item on the home page and company page means, and from where the data is taken. I will navigate to another page other than the home screen and click the "Help" button again to show that it takes me to the same Help screen.		~
UC10	User clicks the "Exit" button: In the bottom right corner of every screen is an exit button. When clicked, the program terminates and the window is closed.	I will navigate to the home page and click the exit button. I will show that this terminates the program. I will reopen the program and navigate to a page other than the home screen, then click the Exit button again to show that it also terminates the program.		

The program passed every portion of every test except for one, as I made an executive decision to change a requirement involving the GUI buttons. In Use Case 7, almost every test passes, as the Home button is visible on every page in the program. However, I decided to display a new window for the “Help” screen, which makes a Home button unnecessary. In Use Case 9, I decided to make the Help button only available on the Home screen, so that a new feature I added could be displayed large enough to read comfortably. If one navigates back to the Home screen, they can easily access the Help screen again.

Everything else passed well. The GUI interface passes each test for every use case, and the navigation buttons, search bar, search functions, price table, sorting functions, home window, company window, are executed as I planned.

I included new features as well, explained in Section 3.4.

## 3.2 Other Testing

Although the Use Cases all passed, there was still more testing to perform. I also had to test the accuracy of the data scrapers, the calculations, the price updates, and the company lists.

Although I planned on performing unit tests, they were not the ideal option, as the APIs I used run simultaneously with the data manipulations and organizations, and I wouldn't be able to test effectively within the system, as I'd have to use APIs to scrape data for those as well.

Instead, I compared the prices, tickers, sorting methods, earnings dates and price variants for the S&P 500 multiple times during the week, after earnings dates, and on days the market was closed. I found a lot of problems and errors at first, described in Section 3.3, but they were eventually resolved through much trial and error. The program functions and reports data accurately as of April 25<sup>th</sup>, 2021.

Testing will need to continue throughout the program's use, as APIs get updated, and as new libraries improve functionality and speed.

## 3.3 Obstacles to Meeting Requirements & Errors Encountered During Tests

There were many problems with accuracy at first. For instance, in the original design document, I'd planned on using the Yahoo Finance API for all scraping procedures, but after implementing the API and encountering earnings errors, I had to test the dates and prices for many companies. It was time consuming, but worth it. I found out that Yahoo Finance has actually reported multiple earnings report dates and times incorrectly. This is a massive problem for options traders who utilize the 24-hour-post-earnings method of options trading, as it would produce inaccurate data. To mediate this issue, I found another website to scrape called ZACKS, which produces accurate earnings dates and times. I also had to configure the time in each scrape to match the U.S. Eastern Time Zone, and found that using the pytz library's timezone function made it easy.

Many issues I encountered were resolved through simple research. If I wasn't sure how to do something or what features a Python library could provide, I could easily find via the web. I solved many bugs just by finding libraries that could perform tasks for me. For instance, from the Python library, Pandas, the DataFrame function saved me a lot of time and energy, as it seamlessly coordinated related data sets. The Pickle library allowed me to store data and update data dictionaries easily.

There was an issue with the speed of program open. To resolve this, I found that functions and library classes/variables like Futures, ThreadPoolExecutor, and pool allowed for multiple tasks within the program to run at once, scraping the data much faster. I also added a console progress bar with a library that made it easy, so that, when running the GUI, the console would update me on progress prior to open.

An issue when it came to data calculation involved making sure the price difference was for the following *market* day, not just the day after (as the stock market is closed for most on weekends

and some holidays), and I found that using the accurate market day features in Yahoo Finance could do just that.

One other problem included some companies not reporting upcoming earnings dates, and only their old ones would append to the dictionary. This didn't allow for efficient sorting. To resolve this, I created a stub date in 2050 to apply to send those companies to the bottom of the list. The list will self-update as earnings report dates are added.

### **3.4 New Requirements/Added Features**

Along with the added console progress bar, I added features as I found them. I used the Tkinter Python library to create the user interface, subsequently learned about many useful features I could simply implement within the windows/frames. I decided to add a stock chart to each company's information page, with the price points at each earnings report date, as I thought a visual representation of price would add another layer of human interaction (This is why I decided to forego one of my former requirements).

I added an installation file and provided a walkthrough of different methods of implementation within the README. I also added a runnable requirements (dependencies) file to make compatibility easier upon download.

Along with the buttons along the side to allow for sorting, I also utilized the DataFrame with Pandas and TreeView in Tkinter to allow for sorting by clicking on the column headings themselves.

I added many updating scrapers to ensure updates upon program open, and the simultaneous running of those updates before program open.

### **3.5 Potential Future Requirements/Features**

In the future, I hope to turn the program into a web app. I would like to recreate the GUI with HTML, as I think that web app GUI design is more advanced and can produce a better result. I'm curious to see if it improves program speed. A web app would allow non-Windows users to access the program as well.

I would also like to make updates run upon instantly, showing real-time stock prices and data instead of just updating upon open.

I am going to continue refining the source code, as I'm sure there are plenty of functions and classes whose efficiency and functionality could be improved. It's likely that there are redundancies, as many of the functions implemented are those I have only learned to utilize recently (as of April 25, 2021), and more learning is necessary.

Although not an application feature, more documentation will be necessary in the near future, along with a code review to ensure the best practices are being utilized throughout the source code.

An easier-to-use installer program is also in the works. I would like to create an installation routine that doesn't require the user to have any Python languages or libraries installed already. I think a better explanation or caption of the daily change columns/features will be necessary as well.

## **4. Design of Program**

### **4.1 Design Overview**

The original design of the source code files were as follows, as defined word-for-word from the Detailed Design Document: “The main file starts the process of running the subroutines. It receives information and organizes data from each module and the README file when needed, and then relays the information to the GUI module. The GUI collects data only from the main file. The main file collects information from the following:

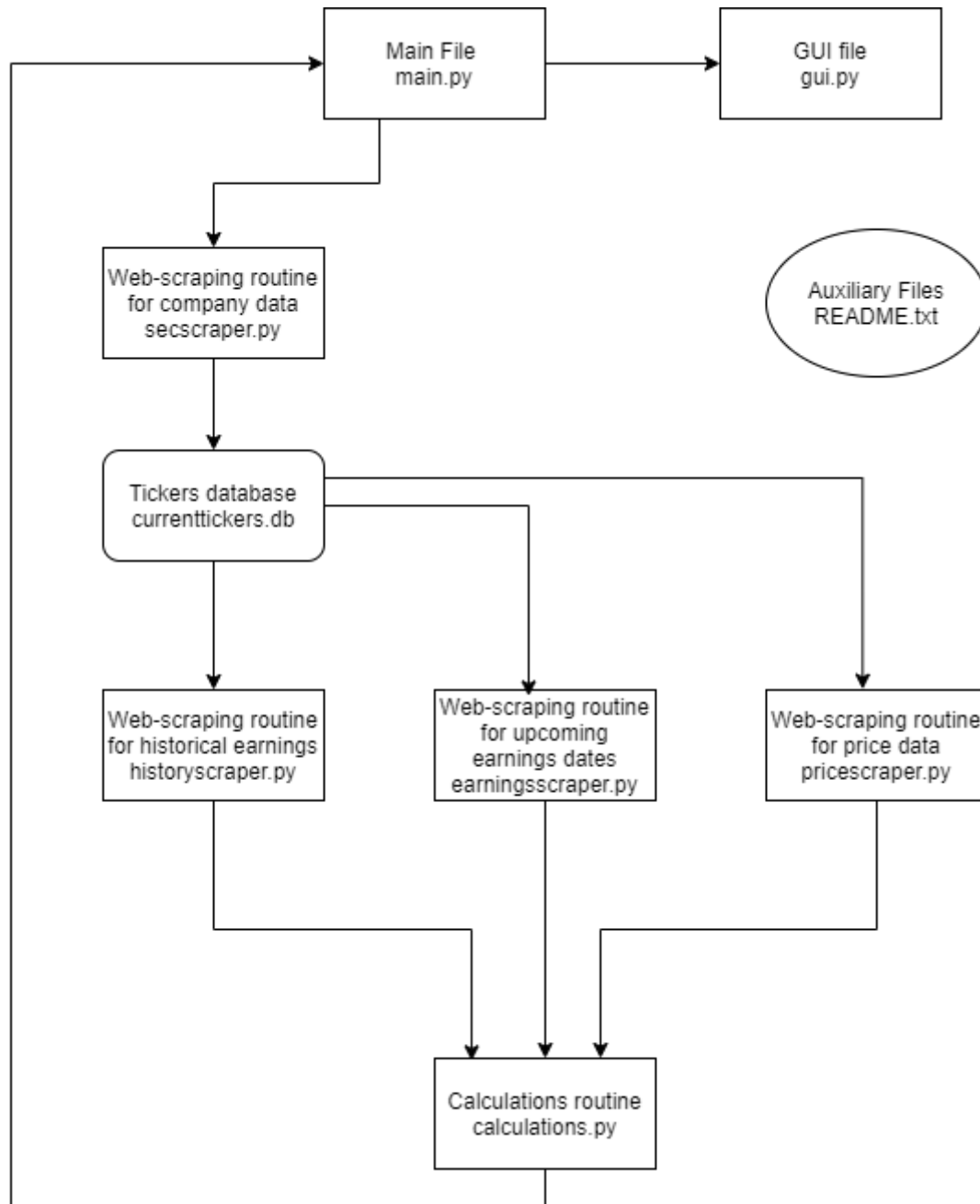
The first web scraping routine collects a current list of the S&P 500, to ensure that no unnecessary company data will be collected for any other routine. It stores this information in a database, which the other subroutines will use to perform their operations.

The historical earnings scraper uses the database as a guideline to scrape the web for the earnings prices throughout history for each company in the database. The upcoming earnings scraper finds the date and time of the companies’ upcoming earnings reports. The price scraper takes the database and finds the updated prices necessary to run the program.

The README file is an auxiliary file not dependent on any other file. It provides a walkthrough of the program and a description for users and developers. Developers can locate the file within the project, and users can request the file’s information from the user interface by clicking the “Help” button.

Data that requires calculation is first sent to a calculations file and then the main module. Other data is sent directly to the main module. In the main file, the data is organized and made ready for use by the GUI when the user requests it.”

The design included multiple scraping files, a calculations file, a README, and a GUI to function in this format:



At the time, I wasn't entirely sure how the functions would perform together, or at what time in the calculations files each scraper would be implemented.

The final design ended up being moderately different, as is described in section 4.2 below.

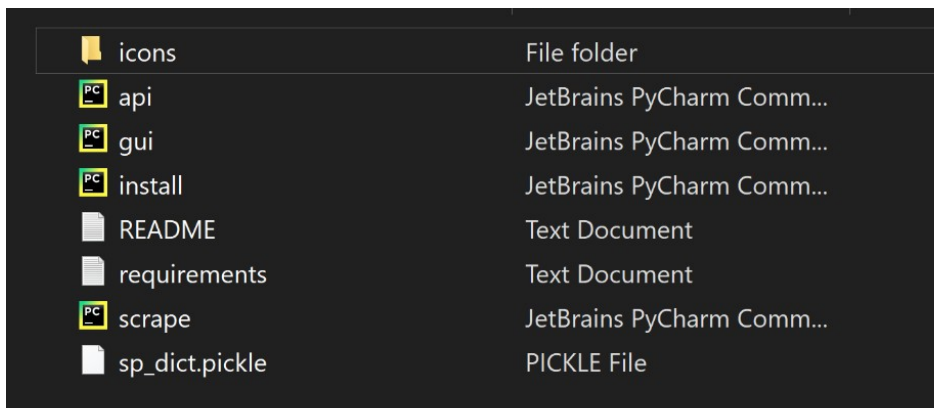
## 4.2 Comparison with Original Design Document

The original design was similar in that it included a README with instructions and a GUI that would be run as the program User Interface. Currently, gui.py runs the program at the executable level.

Some changes had to be made, of course. I found that, when running tasks simultaneously, it was redundant to run multiple scrapers at once, as I could pull different data from the same tables at once with one API. Therefore, instead of multiple scrape files, there is one scrape file called scrape.py. This file defines functions to scrape data that needs to be organized and manipulated in some way to ensure proper placement in the dictionary and table. Another Python file: api.py, runs the scrape file functions and organizes this data in dictionaries/zipped lists of tuples/a pickle file for storage and also performs minor scraping routines in which the data needs little to no manipulation or calculation. That organized data is then sent to the GUI, gui.py.

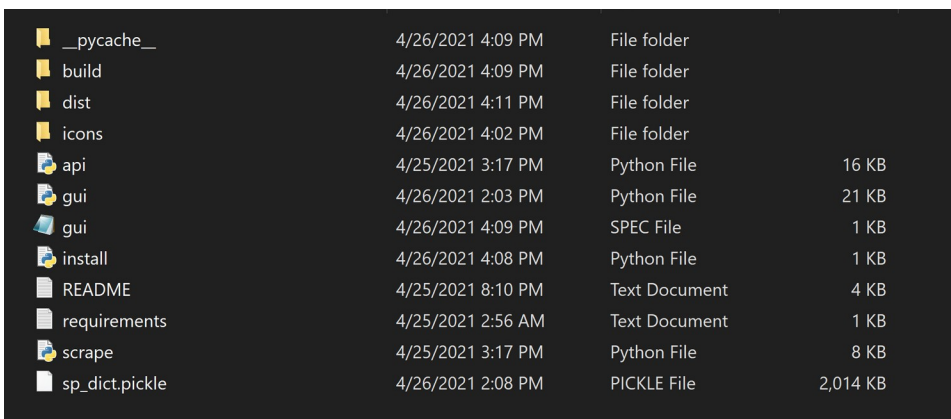
Along with the README, I also added the requirements.txt file to allow for easy dependency checking/downloads and an install.py file to install the program as an executable. I also added an icons folder to add .png icons to the user interface.

Pictured are the items within the current package:



icons	File folder
api	JetBrains PyCharm Comm...
gui	JetBrains PyCharm Comm...
install	JetBrains PyCharm Comm...
README	Text Document
requirements	Text Document
scrape	JetBrains PyCharm Comm...
sp_dict.pickle	PICKLE File

The items in the current package post-installation (after following the installation instructions I provided in the README) are pictured in the following image:



__pycache__	4/26/2021 4:09 PM	File folder	
build	4/26/2021 4:09 PM	File folder	
dist	4/26/2021 4:11 PM	File folder	
icons	4/26/2021 4:02 PM	File folder	
api	4/25/2021 3:17 PM	Python File	16 KB
gui	4/26/2021 2:03 PM	Python File	21 KB
gui	4/26/2021 4:09 PM	SPEC File	1 KB
install	4/26/2021 4:08 PM	Python File	1 KB
README	4/25/2021 8:10 PM	Text Document	4 KB
requirements	4/25/2021 2:56 AM	Text Document	1 KB
scrape	4/25/2021 3:17 PM	Python File	8 KB
sp_dict.pickle	4/26/2021 2:08 PM	PICKLE File	2,014 KB

I'm surprised I decided to change so much, but as I learned more about simultaneously running functions with Futures and utilizing Pickles, I thought it was better this way. In the future, as I add functionality, I will further separate files based on efficiency and understandability.

### 4.3 User Interface Differences

The original idea for the user interface in the Requirements Document:

The wireframe shows a user interface with the following elements:



- Title:** A box labeled "Current S&P 500 Companies" with a home icon and a close (X) icon to its right.
- Table:** A list of 18 entries, each consisting of a grade "AAA" in a rounded box followed by a rounded box containing the text "COMPANY NAME".
- Search:** A text input field labeled "Enter text..." with a blue "SEARCH" button below it.
- Sort:** A green-bordered box labeled "SORT BY:" containing three options: "Upcoming Earnings Date", "Current Price", and "Average % Change Post-Earnings", each in its own green rounded box.
- Buttons:** An orange "HELP" button and a red "EXIT" button at the bottom right.



The next user interface (designed with Adobe Illustrator):

Current S&P 500 Companies

MMM	3M Company	175.00	5.26%	▼
ABT	Abbott Laboratories	0.00	0.27%	▲
ABBV	AbbVie Inc.	0.00	0.00%	▲▼
ABMD	Abiomed	0.00	0.00%	▲▼
ACN	Accenture	0.00	0.00%	▲▼
ATVI	Activision Blizzard	0.00	0.00%	▲▼
ADBE	Adobe Inc.	0.00	0.00%	▲▼
AMD	Advanced Micro Devices	0.00	0.00%	▲▼
AAP	Advance Auto Parts	0.00	0.00%	▲▼
AES	AES Corp	0.00	0.00%	▲▼
AFL	Aflac	0.00	0.00%	▲▼
A	Agilent Technologies	0.00	0.00%	▲▼
APD	Air Products & Chemicals	0.00	0.00%	▲▼
AKAM	Akamai Technologies	0.00	0.00%	▲▼
ALK	Alaska Air Group	0.00	0.00%	▲▼
ALB	Albemarle Corporation	0.00	0.00%	▲▼
ARE	Alexandria Real Estate Equities	0.00	0.00%	▲▼
ALXN	Alexion Pharmaceuticals	0.00	0.00%	▲



Enter text...

SEARCH

SORT BY:

Upcoming Earnings  
Date

Current Price

Average % Change  
Post-Earnings

HELP

EXIT

The final actual user interface created (designed with Tkinter):

Symbol	Company Name	Current Price	Average Change (USD)	Percent Average Change	Upcoming Earnings Date
A	Agilent Technologies	136.68	1.0	1.28	2021-05-20
AAL	American Airlines Group	21.11	0.51	2.41	2021-04-29
AAP	Advance Auto Parts	199.75	2.33	1.64	2021-05-18
AAPL	Apple Inc.	134.32	0.29	1.13	2021-04-28
ABBV	AbbVie Inc.	111.38	1.05	1.26	2021-04-30
ABC	AmerisourceBergen	121.65	1.55	1.85	2021-05-06
ABMD	Abiomed	351.03	1.04	1.03	2021-04-29
ABT	Abbott Laboratories	123.31	0.06	0.0	2050-01-01
ACN	Accenture	291.74	3.06	1.53	2021-06-24
ADBE	Adobe Inc.	515.84	2.49	1.11	2021-06-10
ADI	Analog Devices, Inc.	159.02	2.02	2.02	2021-05-19
ADM	Archer-Daniels-Midland (	59.41	-0.57	-1.21	2021-05-05
ADP	Automatic Data Process	195.86	0.42	0.48	2021-04-28
ADSK	Autodesk Inc.	295.27	1.29	0.9	2021-05-26
AEE	Ameren Corp	84.78	-0.44	-0.65	2021-05-10
AEP	American Electric Power	87.7	-0.36	-0.49	2050-01-01
AES	AES Corp	28.3	-0.26	-1.0	2021-05-06
AFL	Aflac	53.37	-0.11	-0.44	2021-04-28

The user interfaces ended up being very similar, as I stored the icons from the original design I created and uploaded them into Tkinter as buttons. It's a simple design, which I like, but I think it can be improved to look more "grown-up" and add more features/buttons for functionality.

## 4.4 Design Obstacles

The biggest obstacle I faced was not fully understanding Tkinter. I learned a lot throughout the process, but I think a different GUI creator (perhaps using a different language) would look better. On the other hand, there is still a lot I don't know about Tkinter, and I plan on learning more to improve my user interface designs.

Other obstacles included knowing which scrapers to run at which times, trying to parse HTML data with BeautifulSoup, and learning through trial and error how to create tables, DataFrames, pickled lists, zipped tuples, and other functions needed. I'm sure there are redundancies or places in my code where efficiency could be improved, and I'll be constantly learning and testing new findings to make the source code more readable.

Another design obstacle included best practices. I had to figure out how to make proper requests when scraping data and which scraping methods were frowned upon or not allowed in certain cases. In some cases, not utilizing best practices such as requesting headers with the proper browser protocol, produced errors.

Documentation was difficult for me. I wish I had done it along the way while coding and editing the code, as I found that I'd forgotten why I'd done certain things while editing. If I had briefly documented at the beginning, for my own sake, I could have saved time. Instead, I

documented at the end, after finishing. For some functions, I couldn't remember why I'd included them in the first place, and had to go back, reread, research, and test, in order to document correctly.

## **4.5 Python Performance and Packages Utilized**

When it comes to finance, I think Python performs very well. The Pandas and NumPy libraries were especially useful for sorting through finance data. The datetime functions included in Python also proved to be very useful, as accuracy of the program was entirely dependent on date and time relative to other dates and times. I was impressed with the Pickle library for storage, the BeautifulSoup library for data scraping, and the features involving requests\_futures to delegate and allow multiple tasks to perform at once.

All external libraries utilized include python-dateutil, requests, pyinstaller, yfinance, pytz, matplotlib, mplfinance, bs4, ttkthemes, requests\_futures, pandas, dateutil, beautifulsoup4, future, numpy, and StringGenerator. Other resources used included ZACKS.com, Yahoo Finance, and MarketWatch. I researched all in advance to make sure they were well-known and safe within the Python community and on the web. All resources aided the process tremendously.

# **5. Summary of Program and Findings**

## **5.1 Summary of Capstone Project Creation**

To summarize the project, it was, overall, a success. The executable passes almost all use case requirements, performs calculations accurately, scrapes in a timely matter, and is organized and documented so that others may run it. I have already used it in my everyday life, and plan to add more functionality as I learn. I believe the program is truly special, not because it is particularly beautiful or unique, but because it provided a real-world learning experience for me, and automated a significantly time-consuming task I perform nearly weekly. I've already saved time!

## **5.2 What Went Right**

Use cases were passed efficiently. The GUI was relatively simple to create with Tkinter and I believe I have a lot more graphical options to explore. The data scraping processes were simple to implement. Utilizing futures[] and methods of performing multiple updates simultaneously brought the program to a much better speed. The data tables and frames created, along with the dictionaries and pickled lists made calculations a lot easier. I could easily perform data manipulation without going through the slow iteration process, which I'd worried about prior to program design.

The simplicity of adding features, especially in the user interface, pleasantly surprised me. The plotted data graph that looks complex took a line or two of code. Python's many libraries worked well for automation, and their many data structures made storing the data much simpler than I had thought.

I think that designing my Use Cases in advance was the key to the program's success. It streamlined a lot of the pseudocode. I think I utilized web research well for the project, as I found so many libraries and APIs that provided automated solutions to problems I faced.

### 5.3 Lessons Learned

I learned to document as I go! It can not only help another developer after you, but it can help the original programmer in the future upon code reread. I also learned that perhaps the Python language isn't for everything. I think a better GUI could be created in another language/framework, possibly with Python in the backend for data scraping and financial calculation.

One important finding was that the Yahoo Earnings Calendar with the Yahoo Finance API was not always accurate. Even the most well-documented and well-loved APIs and external libraries can breed issues.

Another lesson was the importance of planning programs in advance. This program would have never seen the light of day without the requirements document and the detailed design. Most of all, I learned that computer programmers are *always* learning. There are millions of repositories, data structures libraries, packages, functions, languages, and frameworks just waiting to be utilized.

### 5.4 What's Next For The Project

Next, I plan to share my project to GitHub/forums and let others take a look at it. There are plenty more advanced or knowledgeable than I who can offer insight into program compatibility, simplicity, efficiency, or redundancies.

In the meantime, I plan to learn more about Tkinter and other GUI frameworks to make the interface better-looking. I'd also like to look into more data structures, and perhaps reorganize some of the classes created. I also want to update the icons and installation files to be more seamless.

I would like to eventually turn the program into a web app with live updates and more stock features. I would also like to include all companies in the NYSE and NASDAQ in the near future, as timeliness and efficiency improves.

## 6. Appendix: Resources

Languages and Code Tools: Python 3.9, Pycharm IDE (Developed by JetBrains)

Web Resources: ZACKS.com, MarketWatch.com, Finance.Yahoo.com, Wikipedia.com

Python External/Internal Libraries: Pyinstaller, Tkinter, Dateutil, Requests, Requests\_Futures, YFinance, Pytz, Matplotlib, Mplfinance, BeautifulSoup4, TTKthemes, Pandas, NumPy, Datetime Future, StringGenerator