Dylan Dunsheath Description HW9

**get.py**

This part of the assignment (unlike the php file called **stocks.php**) is what allows us to scrape the data from Yahoo Finance’s most active stocks and store it in our MongoDB database (this part won’t create a page using PHP + HTML to organize a table with the values we collected.

I used four different libraries in my python program: requests; which allows us to send HTTP requests to a specified URL, time; which allows us to measure the start time, rest for 3 minutes, and continuously check the time afterwards to exit after 15 minutes, BeautifulSoup (from bs4) to parse the HTML content, and pymongo to insert into our database…

After declaring these specified libraries, I used time.time() and initialized the value into start\_time so we can use this value to know when 15 minutes have passed. I then initialized a variable called url that takes a string of the website we want to scrape (the Yahoo Market). The MongoClient(‘localhost’, 27017) allows us to create a MongoDB Connection while the client[‘stock\_database’] and db[‘most\_active stocks’] create a database and a collection within that database respectively:

* client = MongoClient('localhost', 27017)
* db = client['stock\_database']
* collection = db['most\_active\_stocks']

One of the crucial parts of this program is the while True loop, which allows us to continuously loop the program and call the getStocks function until 15 minutes have passed. The while True loop does the following:

* gets the elapsed time by subtracting the current time minus the start\_time (which gives us the time in seconds)
* Checks if the elapsed time is 900 seconds or greater: if true, then we have been running this program for 15 minutes and then exit
* If not true, we call the getStocks function (which takes the url provided as a parameter). After this function is done executing, we print a message that the program will “sleep” or wait for 3 minutes (180 seconds) and then repeat until we ran it for 900 seconds.

The getStocks function is another driving force of the program as it is what allows us to scrape the content that’s on the url we provided and update the DB with this data.

* The function begins by sending an HTTP request to the specified url.
* If the status code of the response is 200, it’s successful and begins parsing the HTML content by using BeautifulSoup (it using html = response.text) and ‘html.parser’.
* It then finds the table using soup.find(‘table’, {‘class’:’ W(100%)’}) which will allow us to get the data
* The function also declares a set which has { and } that is best described as a collection that is unordered, unchangeable, and doesn’t allow duplicates (hence why I figured this was best to prevent any potential issues). This was used to get the companies that existed on the website
* The program will then loop through all the rows in the table by using table.find\_all(‘tr’)[1:] 🡪 this will start at the first company rather than also getting the headers
  + For each row, I also used find\_all on the ‘td’ and then used indexes: 0, 1, 2, 3, and 5 and initialized the specific header as a variable (symbol, name, price, change, and volume respectively) along with .text to get the value from the table
  + I also used .add(symbol) on the set to add the symbols on the site
* I then initialized a dictionary which has the headers that were previously mentioned as the ‘key’ of the dictionary and the table data that was extracted as its corresponding value.
* I constructed an if-else condition after the dictionary: one where the symbol does not exist in the collection and another where it does. If the symbol does not exist; then it needs to be added to the collection as it is not in it. Otherwise, I did an update on the price, change, and volume by using what is known as $set and the update\_criterai as the symbol. update\_one is what allows us to update things within the collection.
* After all the data was inserted/updated, I used my dictionary and collection.find() to go through my collection and delete any content that isn’t on the website anymore. I began doing it by extracting the vlue from the doc dictionary (ie: symbol\_in\_db = doc[‘Symbol’)) and checked if the symbol\_in\_db was not in symbols\_on\_website set (which will be empty when we call the function it’s in so we don’t need to worry about this case). If it’s not in the set, it clearly is not on the website so we can delete it from the collection.
* If the program initially fails to get a 200 status code via the HTTP request, it tries again and follows the steps previously mentioned.
  + If it fails again, It prints a message that the retry failed and we are exiting and then uses exit() to terminate the program

As previously mentioned, the while loop continuously runs (assuming we haven’t exit the program). After we call the getStocks, we sleep for 3 minutes, get another elapsed time and check if it was 15 minutes have passed and exit if so. It repeats until the program exits

**stocks.php**

This file is what allows us to generate a webpage about the information that we collected from our python program of the “Most Active Stocks.”

PHP Section

In this program, I used header(“refresh: 180”) so the webpage will refresh every 3 minute to keep the content up to date around the time it updates the stocks from our collection (it won’t be exactly the time). The vendor/autoload.hp uses/includes the composer autoloader to autoload classes from the vendor directory. Composer essentially is a tool for dependency management and we use it to manage libraries on our php projects.

Afterwards, we create a new instance of the new MongoDB\Client class and establish a connection to our MongoDB server. This won’t work UNLESS you have the PHP Driver installed. Ie:

* $client = new MongoDB\Client();
* $database = $client -> stock\_database;
* $collection = $database -> most\_active\_stocks;
  + This makes a connection to the MongoDB server 🡪 selects the stock\_database from MongoDB 🡪selects most\_active\_stocks collection from stock\_database database.

Afterwards, we do $result = $collection->find([]) to execute a query on the most\_active\_stocks collection and retrieve all documents

HTML Section

This is what allows us to display the webpage and design the layout, including: Title (called Active Stocks), headers:

* H1: Most Active Stocks
* H2: Created by Dylan Dunsheath (CS-288-006)
* H3: Source: Click here (as a link to the site)

After adding the various headers, I used a style tag to stylize the table, the sortable headers (which is a class and used the dot [.] operator), and the th, td tags. The table takes an id called “stocktable” with 5 headers: Index, Symbol, Name, Price, Change, and Volume (which as mentioned, uses a class called “sortable.” Within the tbody called (table body), I initialized within php tags a index variable which is initialized to 1.

Afterwards, I declared a foreach loop that loops over result as “doc” and then adds a table row for all documents that exist in the collection (along with adding the index as a portion of data in the table. There is then another loop that gets the key-value pair for all documents. Ignoring the \_id for all documents in the collection, we then add the table data for the other headers in the table and after the outer loop, close the table row for that specific column we are on and then increment the index variable. After the table tag, we use a script tag that uses the jquery tablesorter that sorts the table we implemented which has an id of stockTable so if you click on the headers, it sorts the values based on that header in either ascending or descending order.