RESTful API and User Documentation

[**Collection Management**](#_vgyd5rpokigy) **2**

[Create a database](#_f57pa4fbe6e0) 2

[**Document Manipulation**](#_cdywvx7dxikp) **3**

[Insert](#_4z4uvj3643gf) 3

[Read](#_ljirb5zd3ywa) 3

[Update](#_xikx8z7lwnvo) 4

[Delete](#_uxi5841b4oke) 5

[**Document Retrieval**](#_ckwld646h6y4) **6**

[50-Day Simple Moving Average](#_xh4hec6zvh0h) 6

[Industry](#_dyfc1cd2ma7l) 6

[Aggregation Pipeline](#_l226x9ah17tj) 7

[**Advanced Programming Project**](#_dxl0yclu652) **8**

[CRUD](#_7f6vgpxiu3b0) 8

[Advanced Querying (Ticker List)](#_9ece1izd91hw) 9

[Advanced Querying (Top 5 Stock by Market Cap)](#_cfpz5dg9br4m) 10

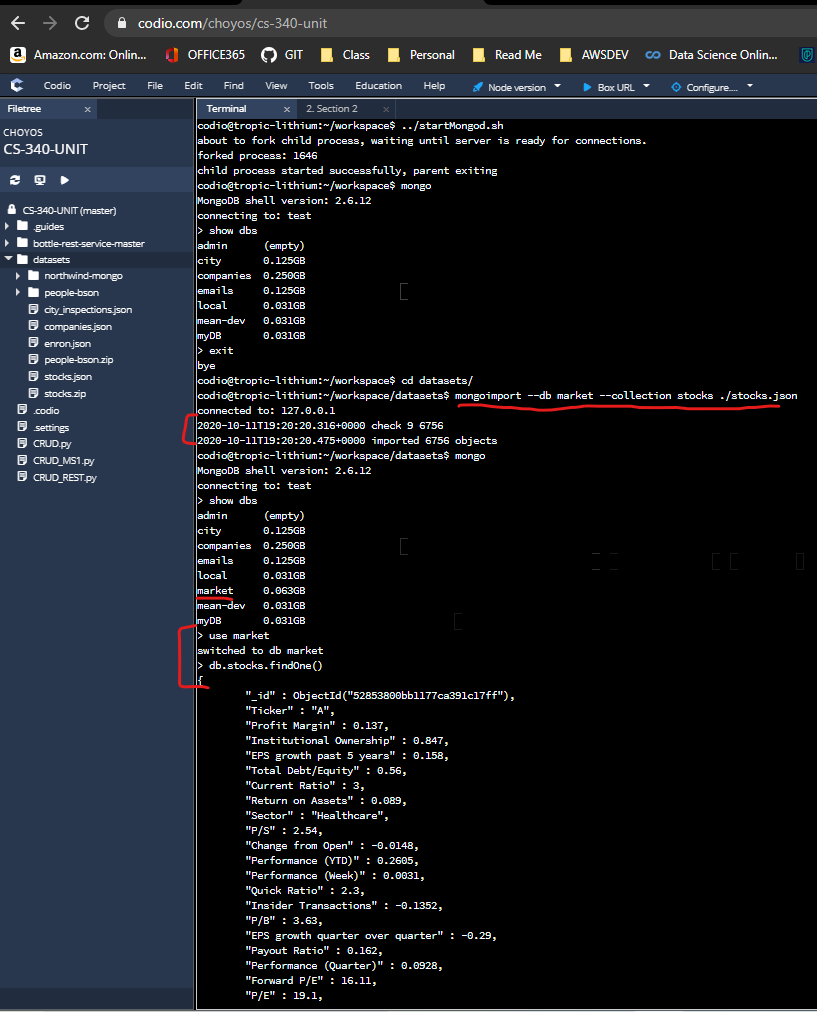
[Row 10 URI](#_gpus0sx9h4hb) **12**

### Collection Management

#### Create a database

‘mongoimport --db market --collection stocks ./stocks.json’

This command is responsible for creating a database called ‘market’ and then subsequently inserting all documents inside of ‘stocks.json’ into the collection called ‘stocks’. We can verify that the collection was loaded successfully by using ‘db.stocks.findOne()’. This shows us an example file.



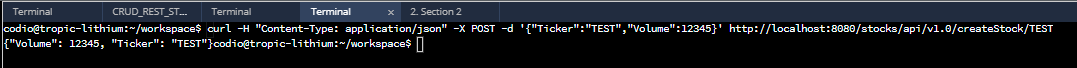
### Document Manipulation

#### Insert

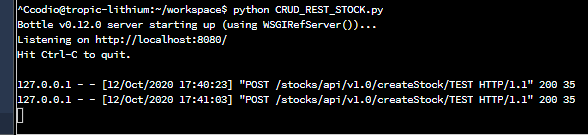
Create:

(Row 3 URI)

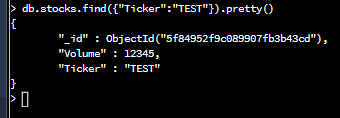
This screenshot below shows the creation of a document in mongoDB using a RESTful web interface. Two key things to understand, the data passed in the POST is ‘{“Ticker”:”TEST”,”Volume”:12345}’ and the URI used to pass the data is ‘http://localhost:8080/stocks/api/v1.0/createStock/TEST’.



This screenshot shows the post succeeded with a 200 response (OK).



This screenshot is used to verify that the TEST ticker we passed successfully found its way to the database.



#### Read

Read:

(Row 4 URI)

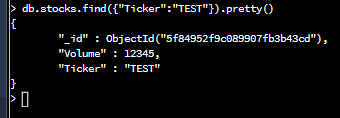
This screenshot below shows the creation of a document in mongoDB using a RESTful web interface. The key thing to understand is that the URI utilizes the last component to find the document. ‘http://localhost:8080/stocks/api/v1.0/getStock/TEST’ You can see that we use ‘TEST’ to look for this document.



In the lower half, you can see some example ‘getStock/TEST’ commands working successfully with a 200 response (OK).



This screenshot is used to verify that the TEST ticker we searched for exists in the database.



#### Update

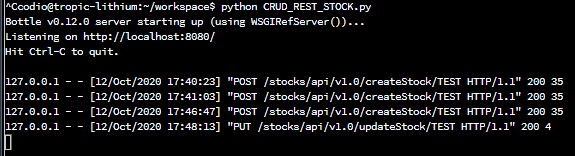
Update:

(Row 5 URI)

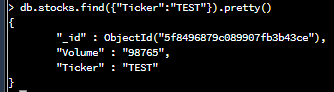
This screenshot is showing how to update a document. Two key components are in the URI you can see which ticker is trying to be updated, that is ‘TEST’, and the volume being passed in the PUT contains a volume amount it’s going to update which is 98765.



This screenshot shows we have a successful PUT at the bottom with a 200 response (OK).



This screenshot shows the change has been made and the volume is now 98765.



#### Delete

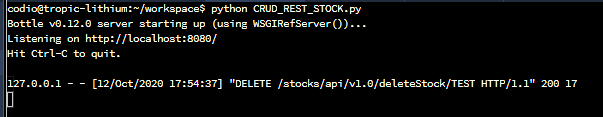
Delete:

(Row 6 URI)

This screenshot shows which stock will be deleted based on the URI ‘TEST’.



This screenshot shows we have a successful DELETE with a 200 response (OK).



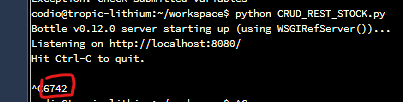
This now shows that when we look for the TEST ticker, the database no longer has the entry.



### Document Retrieval

#### 50-Day Simple Moving Average

This screenshot shows an output for an advanced query looking for the number of stock entries with a 50-day simple moving average between -1 and 5.

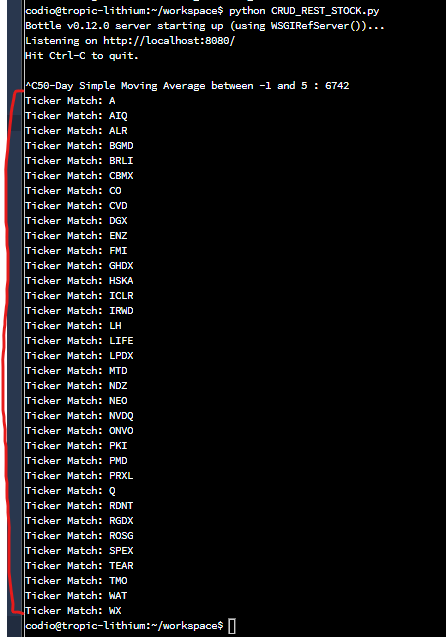


This screenshot shows verification of the result from the python command.

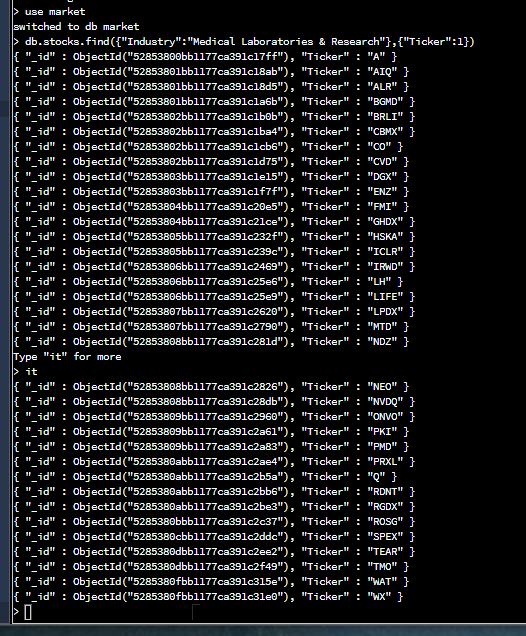


#### Industry

These screenshots show the ability to find ticker symbols based on a particular industry. Here we see the results from python searching on industry “Medical Laboratories & Research”.

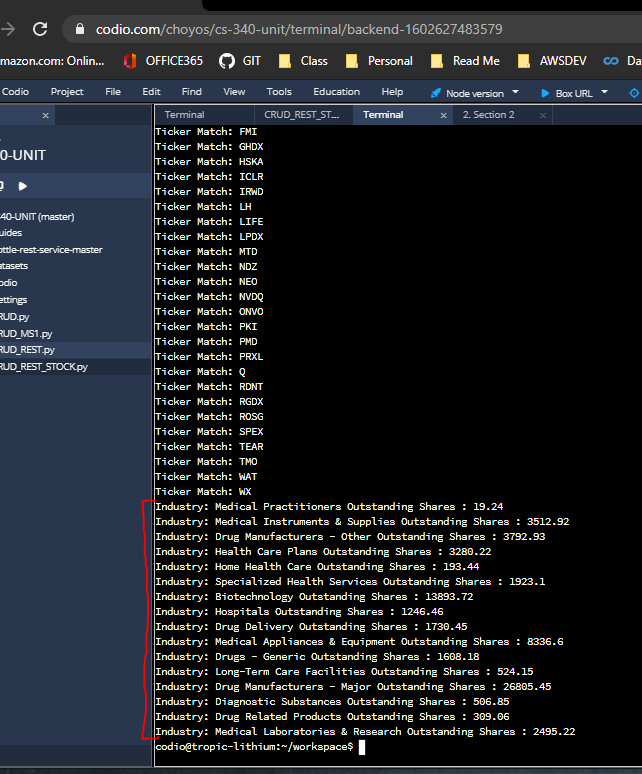


Here we see verification of the results from python by using a mongoDB query. We look for all matches of the industry “Medical Laboratories & Research” and we see the same ticker results.

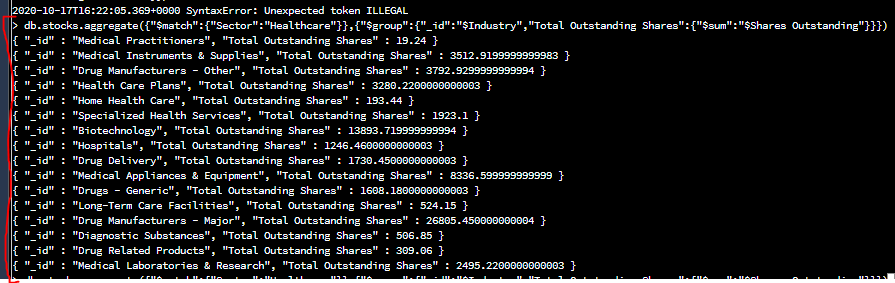


#### Aggregation Pipeline

In the screenshot below we see outstanding shares as executed by a python script. It looks for matches in a sector, such as “Healthcare” and then groups the information based on industry, such as “Health Care Plans”. It then sums up all outstanding shares and provides a total for that group.



Here the results are verified using a mongoDB query.



### Advanced Programming Project

#### CRUD

CRUD are the standard Create, Read, Update, and Delete operations for a database. Please see document hyperlinks below which take you to the relevant examples.

Create - Please see [Document Manipulation Insert](#_4z4uvj3643gf)

Read - Please see [Document Manipulation Read](#_ljirb5zd3ywa)

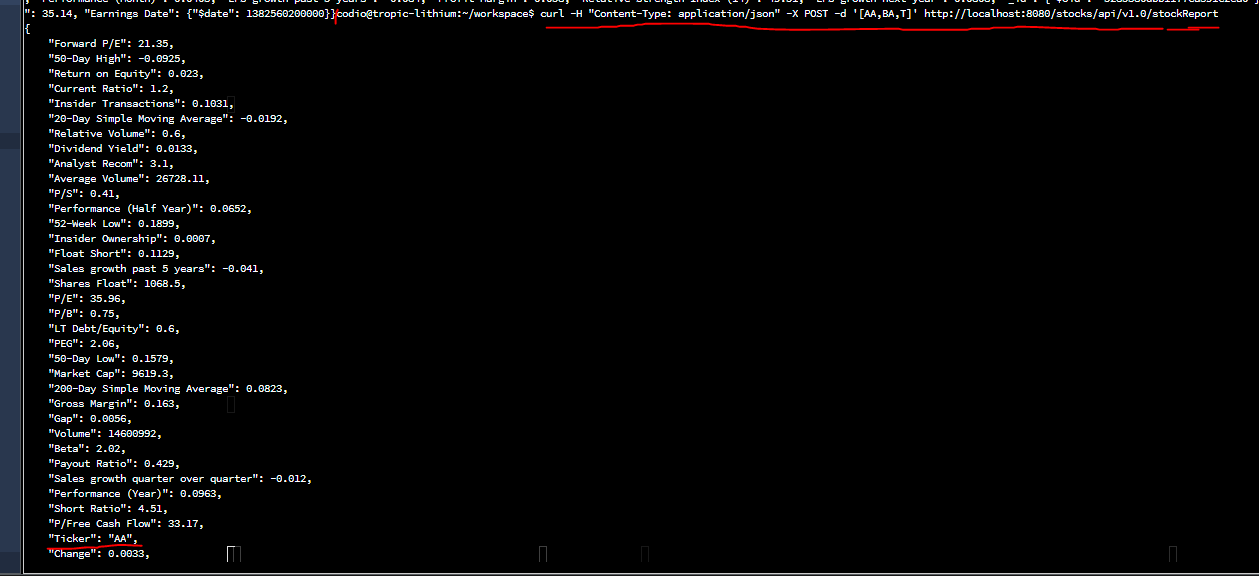
Update - Please see [Document Manipulation Update](#_xikx8z7lwnvo)

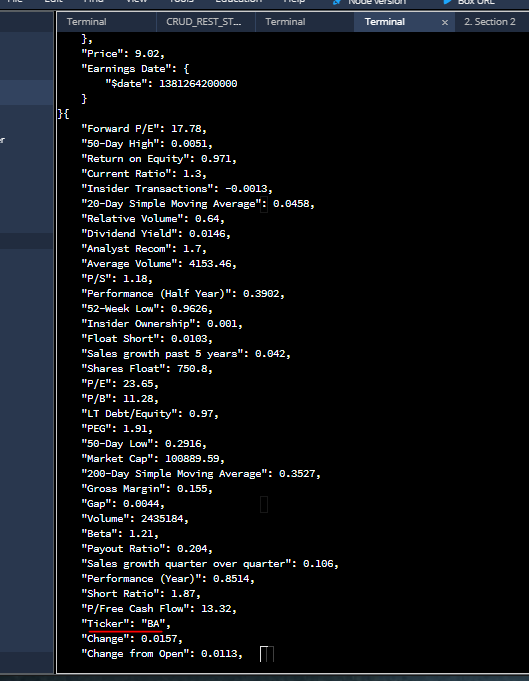
Delete - Please see [Document Manipulation Delete](#_uxi5841b4oke)

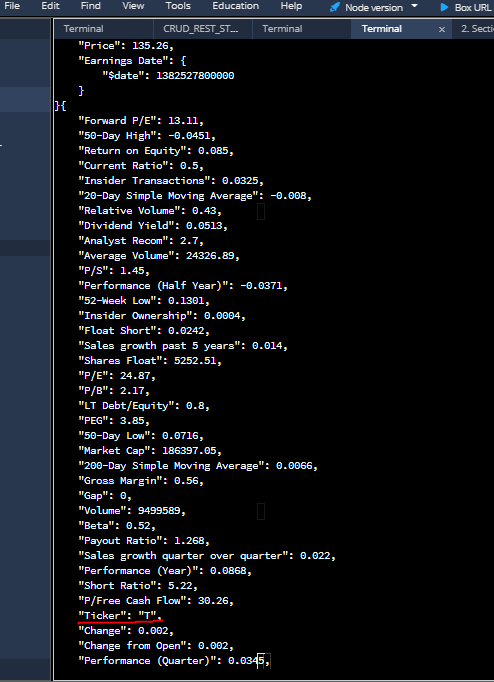
#### Advanced Querying (Ticker List)

In these screenshots we can see the request for a stock report with the following ticker symbols “AA”, “BA”, and “T”. You can see all results provided below for all three tickers.

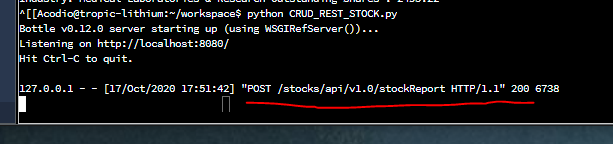
(Row 8 URI)







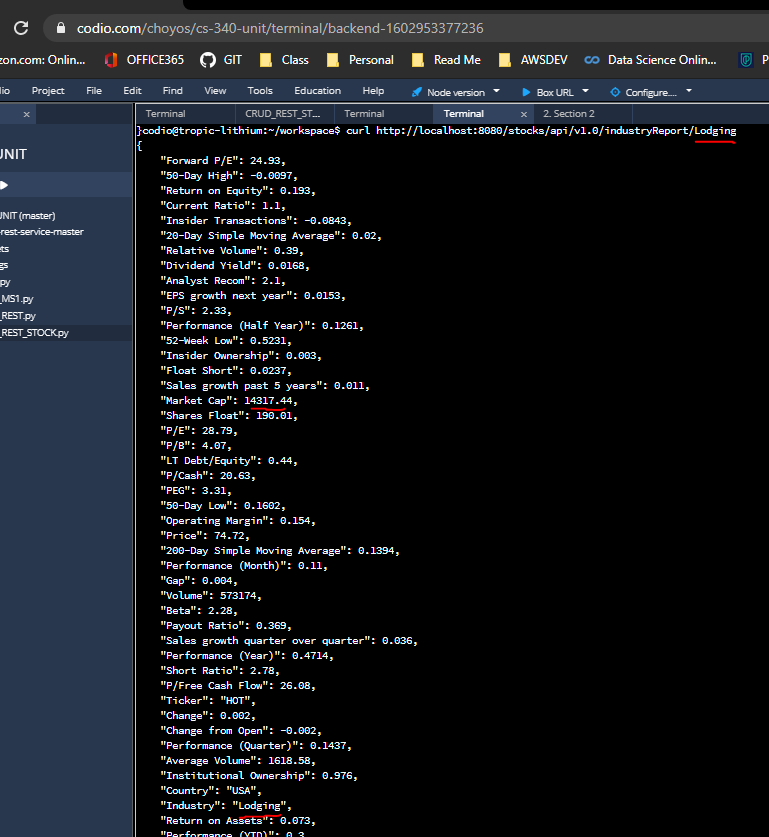
This shows a successful message of 200 (OK) when we execute this HTTP request.

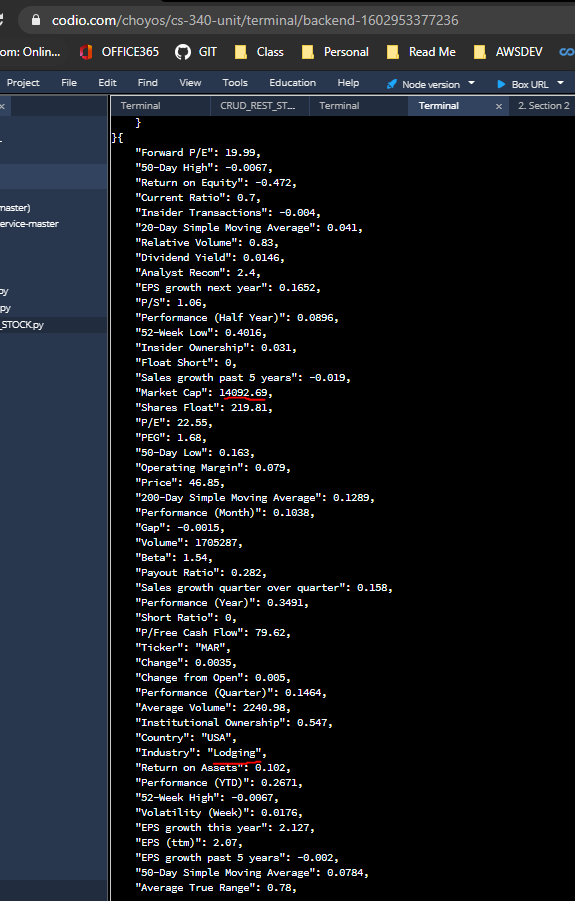


#### Advanced Querying (Top 5 Stock by Market Cap)

This screenshot shows a full report of each ticker that matches a particular industry. In this example we search for all stocks that match the ‘Lodging’ industry. It will find the top 5 in that industry and report them based on market capitalization. Only the first two results are shown so as not to clutter the document.

(Row 9 URI)

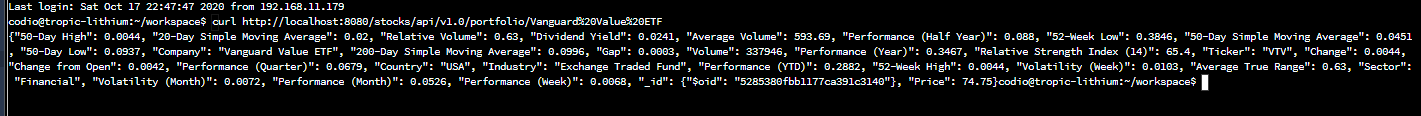




### Row 10 URI

This screenshot shows how passing through the URI ‘Vanguard Value ETF’ we can find the stock’s information. Most other representations use the ‘Ticker’ value or ‘Industry’ to search. Please note it must match HTTP form and the spaces are converted to %20.

(All other URI are provided in previous sections)



This displays a successful message 200 (OK) of the earlier query.



Finally here we show verification of the result by showing it through a mongoDB query separately.

