A RESTful API and User Documentation for a Financial Service

As a newly hired software developer for a startup business, my job is to assess the company's needs and design and implement a stock market information service which is easily understood, accessible and flexible. For this reason, I've chosen to implement a RESTful application interface (API) architectural style. REST stands for representational state transfer which is a highly accessible design that takes advantage of existing protocols, particularly HTTP.

Additionally, I've chosen to use a distributed database (NoSQL/non-relational) called MongoDB since the company has professed to have future goals for securities other than company stock and also because of its driver API's versatility in various languages. For this project I've chosen Python, version 2.7.6. The RESTful API is a part of popular application server frameworks; thus developers only need to test and code uniform resource identifiers (URI's) and locators (URLs) which are paths that allow for high level functionality using an HTTP connection. For the project we will use client side URL's or CURLS.

Collection Management

The first step in collection management is to create a database using the Mongo import tool. Here I create a database which holds collections of data. Note that we can have more than one collection in our database. Here I've taken the data provided me by the company and inserted it into an active directory named 'datasets' with a database named 'market' with a collection named 'stocks'. The terminal then confirms that 6,756 objects (documents) were successfully imported. Each document stores a set of data for a particular company.

```
codio@opinion-trinity:~/workspace$ ls
create.py  datasets  delete.py  hello.py  read2.py  read.py  rest_server.py  test.py  time.py  update.py
codio@opinion-trinity:~/workspace$ cd datasets
codio@opinion-trinity:~/workspace/datasets$ mongoimport --db market --collection stocks ./stocks.json
connected to: 127.0.0.1
2020-10-09T15:04:04.924+0000 check 9 6756
2020-10-09T15:04:05.345+0000 imported 6756 objects
```

After assessing the data and the functionality I would like to implement, I decided to create several indexes for the database which will take advantage of what a distributed database is capable of. Traditional SQL relational databases hold your data in a table with rows and columns which require you to perform table-scans to store, retrieve, update, and delete the data within. Table scans require you to search the entire database which becomes tedious and slow as your database grows.

A NoSQL distributed database uses rows only. This ability allows you to index fields so that when searches are performed on the indexed field, only that field is looked at (by using a pointer). This option will yield higher performance and efficiency as your database grows. For example, say you have ten thousand documents. Indexed fields have the power to limit a search of the entire database by more than one factor so that, for example, only twenty documents are accessed when doing a search- rather than the entire database. Indexes are not for all situations. Consider carefully whether you need access to a few, or all documents to perform some function.

For example, a typical query (table scan) on the stocks collection for industries titled “Medical Laboratories & Research” takes an average of 8 milliseconds. This number will grow

as your collection does.

```
> db.stocks.find({"Industry" : "Medical Laboratories & Research"}, {"Ticker" : 1, "_id" : 0}).explain()
{
        "cursor" : "BasicCursor",
        "isMultiKey" : false,
        "n" : 35,
        "nscannedObjects" : 6756,
        "nscanned" : 6756,
        "nscannedObjectsAllPlans" : 6756,
        "nscannedAllPlans" : 6756,
        "scanAndOrder" : false,
        "indexOnly" : false,
        "nYields" : 52,
        "nChunkSkips" : 0,
        "millis" : 8,
        "server" : "opinion-trinity:27017",
        "filterSet" : false
}
```

After an index was created for the key field “Industries” this search now only takes under one millisecond. Therefore, MongoDB was a great choice for this industry, as the database promises to grow rather large due to its founder’s aspirations.

```
> db.stocks.ensureIndex({"Industry" : 1})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
> db.stocks.find({"Industry" : "Medical Laboratories & Research"}, {"Ticker" : 1, "_id" : 0}).explain().
millis
0
```

This type of index only indexes one field, but you can index as many fields as you like. I made several more single-field indexes to assist in the efficiency of programs which will be described later. Here I make a “Sector” index. Note that you can give the indexes unique names (otherwise the names defaults to “NAME_1”). Also note that the ‘1’ is ascending, you can also use -1 for descending indexes.

```
> db.stocks.ensureIndex({"Sector" : 1}, {"name" : "sector"})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 3,
        "numIndexesAfter" : 4,
        "ok" : 1
}
```

The field ‘ticker’ is an acronym for company names in the collection. I decided to make the field “Ticker” a unique index. What this does is ensure that no duplicate ‘tickers’ can be added to the database, much like a unique id. Currently when a new document is created it’s given an id automatically, but this doesn’t ensure that it’s1/ unique, as every id created like this is unique in itself. Thus making the ‘ticker’ unique ensures not only that duplicates can’t be made, but that creates, reads, updates and deletes (CRUD functionality) that rely on the ‘ticker’ id do not become dysfunctional or irrelevant due to duplicate documents. Updates, for example, might only work on one document- making feedback to the user possibly flawed. Note that I tried to give this index a ‘name’ of my creation, but it did not work. Programming is a continuous learning experience and I learned here that unique indexes cannot be named.

```
> db.stocks.ensureIndex({"Ticker" : 1}, {"unique" : true}, {"name" : "tickers"})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 2,
        "numIndexesAfter" : 3,
        "ok" : 1
}
```

Lastly, I created a compound index for a program I made which can find a document that matches a user defined industry and returns a list of ticker symbols (later described). A compound index will index more than one field. Now we can limit the search of the entire database (6,756 documents) by more than one factor, selecting first only the matching user’s

industry, and then selecting only the ticker fields of those industries to present to the user with no need to scan the entire document for the tickers.

```
> db.stocks.ensureIndex({"Ticker" : 1, "Industry" : 1})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 4,
        "numIndexesAfter" : 5,
        "ok" : 1
}
```

```
"n" : 35,
"nscannedObjects" : 6756,
"nscanned" : 6756,
"nscannedObjectsAllPlans" : 6756,
"nscannedAllPlans" : 6756,
```

A table scan. Here, "n" is found objects after a scan.

```
"n" : 35,
"nscannedObjects" : 35,
"nscanned" : 35,
"nscannedObjectsAllPlans" : 35,
"nscannedAllPlans" : 35,
```

An indexed scan. Here, scanned objects is only 35.

Document Manipulation

Now on to the fun stuff. The following programs were designed to be modular, or in other words, reusable as well as improvable. They are reusable in that entries can be inputted by users in real-time, they do not contain hard-coded user options. These program's abilities are only limited by the imagination. In any of the programs you could add or remove key fields (meaning the first entry in a document's listing- which are all presented as key/values and their possible subdocument key/values and which we can also reach onto using MongoDB), and any of these key fields could be a user defined entry. In this way, the possibility for improvements, or more complex reads, writes, and other database manipulations are endless.

```
"Company" : "Abaxis Inc.",
"Gap" : -0.0014,
"Relative Volume" : 0.23,
```

An example of only 3 of the 66 possible key/values available in each document

These following programs are non-web framework based, in that they do not use the REST API (but they easily could). These programs could work internally in the organization.

This program takes user defined input and creates a new document to add to the ‘stocks’ collection in the ‘market’ database. When the program executes, it will ask the user to enter data in the form {“key” : “value”} with the entries separated by commas. This data could be copy pasted or it could also be formatted, using a program, previous to entry.

```
import json
from bson import json_util
from bson import errors
from bson.json_util import dumps, loads
from pymongo import MongoClient
from pymongo import errors

connection = MongoClient('localhost', 27017)
db = connection['market']
collection = db['stocks']

def insert_doc(document):
  try:
    myInsertResult = collection.insert_one(document)
    print("Document created!")
    return True
  except errors.DuplicateKeyError as e:
    print("Duplicated key error")
    return False
  except errors.WriteError as we:
    print("MongoDB returned error message")
    return False
  except errors.WriteConcernError as wce:
    print("MongoDB returned error message")
    return False

def create_doc():

  #create new stock document
  print('Please enter data in the form: {"key" : "value"} with "key:value" entries separated by commas')

  try:
    stockDoc = json.loads(raw_input())
  except ValueError:
    print("ValueError: wrongly formatted doc!")
    return "Error occured"

  #insert document
  insertResult = insert_doc(stockDoc)

  #write true if inserted, false if not
  print(insertResult)

create_doc()
```

When the document is created successfully, we get the result “Document created! True”. The following entry was one fabricated by me and can easily be removed using the delete program.

```
codio@opinion-trinity:~$ cd workspace
codio@opinion-trinity:~/workspace$ python stocks_create.py
Please enter data in the form: {"key" : "value"} with "key:value" entries separated by commas
{"Ticker" : "JK","Profit Margin" : -0.075,"Institutional Ownership" : 0.003,"EPS growth past 5 y
ears" : 0.242,"Total Debt/Equity" : 7.46,"Current Ratio" : 0.7,"Return on Assets" : -0.074,"Sect
or" : "Services","P/S" : 0.06,"Change from Open" : 0.0205,"Performance (YTD)" : -0.9353,"Perform
ance (Week)" : -0.137,"Quick Ratio" : 0.7,"P/B" : 0.82,"EPS growth quarter over quarter" : 0.119
,"Performance (Quarter)" : -0.767,"200-Day Simple Moving Average" : -0.7382,"Shares Outstanding"
 : 0.99,"52-Week High" : -0.9308,"P/Cash" : 1,"Change" : 0.0687,"Analyst Recom" : 3,"Volatility
(Week)" : 0.0996,"Country" : "USA","Return on Equity" : 6.889,"50-Day Low" : 0.1318,"Price" : 2.
49,"50-Day High" : -0.7658,"Return on Investment" : -0.011,"Shares Float" : 3.09,"Industry" : "M
anagement Services","Beta" : -2.5,"Sales growth quarter over quarter" : 9.286,"Operating Margin"
 : -0.017,"EPS (ttm)" : -5.97,"52-Week Low" : 0.1318,"Average True Range" : 0.37,"Company" : "Je
ssicus Kilbourneous Inc.","Gap" : 0.0472,"Relative Volume" : 3.24,"Volatility (Month)" : 0.0714,
"Market Cap" : 2.3,"Volume" : 46666,"Gross Margin" : 0.292,"Performance (Half Year)" : -0.7376,"
Relative Strength Index (14)" : 22.46,"Insider Ownership" : 0.071,"20-Day Simple Moving Average"
 : -0.4444,"Performance (Month)" : -0.6331,"LT Debt/Equity" : 4.11,"Average Volume" : 15.83,"EPS
 growth this year" : 0.791,"50-Day Simple Moving Average" : -0.589}
Document created!
True
```

Another way to check the for the successful creation of a program is to search for it in the actual database by doing a query “db.stocks.find ({“Ticker” : “JK”})

```
> db.stocks.find({"Ticker" : "JK"})
{ "_id" : ObjectId("5f8da0e0158d440694cd6a02"), "50-Day High" : -0.7658, "Return on Equity" : 6
.889, "Current Ratio" : 0.7, "20-Day Simple Moving Average" : -0.4444, "Relative Volume" : 3.24
, "Analyst Recom" : 3, "Average Volume" : 15.83, "P/S" : 0.06, "Performance (Half Year)" : -0.7
376, "52-Week Low" : 0.1318, "Insider Ownership" : 0.071, "Market Cap" : 2.3, "Shares Float" :
3.09, "50-Day Simple Moving Average" : -0.589, "P/B" : 0.82, "LT Debt/Equity" : 4.11, "50-Day L
ow" : 0.1318, "Operating Margin" : -0.017, "Price" : 2.49, "200-Day Simple Moving Average" : -0
.7382, "Performance (Month)" : -0.6331, "Gap" : 0.0472, "Volume" : 46666, "Beta" : -2.5, "P/Cas
h" : 1, "Sales growth quarter over quarter" : 9.286, "EPS growth this year" : 0.791, "Relative
Strength Index (14)" : 22.46, "Ticker" : "JK", "Change" : 0.0687, "Change from Open" : 0.0205,
"Performance (Quarter)" : -0.767, "Institutional Ownership" : 0.003, "Country" : "USA", "Indust
ry" : "Management Services", "Return on Assets" : -0.074, "Performance (YTD)" : -0.9353, "52-We
ek High" : -0.9308, "Volatility (Week)" : 0.0996, "EPS (ttm)" : -5.97, "EPS growth past 5 years
" : 0.242, "Average True Range" : 0.37, "Sector" : "Services", "Shares Outstanding" : 0.99, "Qu
ick Ratio" : 0.7, "Total Debt/Equity" : 7.46, "Return on Investment" : -0.011, "Volatility (Mon
th)" : 0.0714, "EPS growth quarter over quarter" : 0.119, "Gross Margin" : 0.292, "Profit Margi
n" : -0.075, "Performance (Week)" : -0.137, "Company" : "Jessicus Kilbourneous Inc." }
>
```

If there is already an entry ‘ticker’ with the acronym “JK” we get a duplicate error:

```
codio@opinion-trinity:~/workspace$ python stocks_create.py
Please enter data in the form: {"key" : "value"} with "key:value" entries separated by commas
{"Ticker" : "JK","Profit Margin" : -0.075,"Institutional Ownership" : 0.003,"EPS growth past 5 yea
rs" : 0.242,"Total Debt/Equity" : 7.46,"Current Ratio" : 0.7,"Return on Assets" : -0.074,"Sector"
: "Services","P/S" : 0.06,"Change from Open" : 0.0205,"Performance (YTD)" : -0.9353,"Performance (
Week)" : -0.137,"Quick Ratio" : 0.7,"P/B" : 0.82,"EPS growth quarter over quarter" : 0.119,"Perfor
mance (Quarter)" : -0.767,"200-Day Simple Moving Average" : -0.7382,"Shares Outstanding" : 0.99,"5
2-Week High" : -0.9308,"P/Cash" : 1,"Change" : 0.0687,"Analyst Recom" : 3,"Volatility (Week)" : 0.
0996,"Country" : "USA","Return on Equity" : 6.889,"50-Day Low" : 0.1318,"Price" : 2.49,"50-Day Hig
h" : -0.7658,"Return on Investment" : -0.011,"Shares Float" : 3.09,"Industry" : "Management Servic
es","Beta" : -2.5,"Sales growth quarter over quarter" : 9.286,"Operating Margin" : -0.017,"EPS (tt
m)" : -5.97,"52-Week Low" : 0.1318,"Average True Range" : 0.37,"Company" : "Jessicus Kilbourneous
Inc.","Gap" : 0.0472,"Relative Volume" : 3.24,"Volatility (Month)" : 0.0714,"Market Cap" : 2.3,"Vo
lume" : 46666,"Gross Margin" : 0.292,"Performance (Half Year)" : -0.7376,"Relative Strength Index
(14)" : 22.46,"Insider Ownership" : 0.071,"20-Day Simple Moving Average" : -0.4444,"Performance (M
onth)" : -0.6331,"LT Debt/Equity" : 4.11,"Average Volume" : 15.83,"EPS growth this year" : 0.791,"
50-Day Simple Moving Average" : -0.589}
Duplicated key error
False
codio@opinion-trinity:~/workspace$
```

Finally, if the user enters invalid input, we get an error that the document has a bad format:

```
IndentationError: unexpected unindent
codio@opinion-trinity:~/workspace$ python stocks_create.py
Please enter data in the form: {"key" : "value"} with "key:value" entries separated by commas
dfdfd
ValueError: wrongly formatted doc!
codio@opinion-trinity:~/workspace$
```

This next program will accept a user defined ‘ticker’ to find a file, and update the ‘volume’ key with a new user defined value.

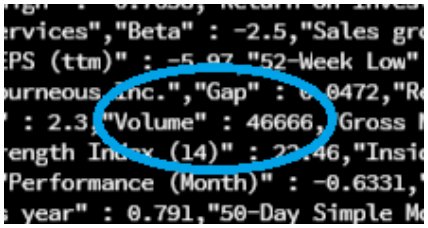
```
1   import json
2   from bson import json_util
3   from bson.json_util import dumps
4   from pymongo import MongoClient
5   from pymongo import errors
6
7   connection = MongoClient('localhost', 27017)
8   db = connection['market']
9   collection = db['stocks']
10
```

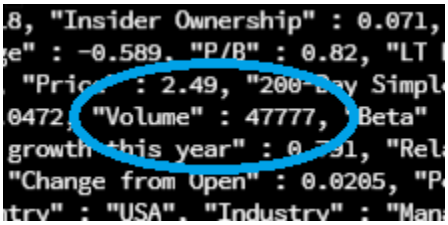
```
def update_document(query, newMod):
  try:
    myUpdateResult = collection.update_one(query, newMod)
    return myUpdateResult
  except errors.DuplicateKeyError as e:
    print("Duplicated key error", e)
    return False
  except errors.WriteError as we:
    print("MongoDB returned error message", we)
    return False
  except errors.WriteConcernError as wce:
    print("MongoDB returned error message", wce)
    return False
  except errors.PyMongoError as pm:
    print("MongoDB returned error message", pm)
    abort(400, str(pm))
    return

def modify_doc():

  #request formatted data for deletion
  print('Please enter document to be updated in the form: {"key" : "value"}')

  #take variable for query
  try:
    myQuery = json.loads(raw_input())
  #return error if badly formatted data
  except ValueError:
    print("ValueError: wrongly formatted doc!")
    return "Error occured"

  #take variable to be modified and updated
  print('Please enter data to be updated in the form: {"key" : "value"}')
  try:
    update = json.loads(raw_input())
  #return error if badly formatted data
  except ValueError:
    print("ValueError: wrongly formatted doc!")
    return "Error occured"
  except TypeError:
    print("ValueError: wrongly formatted doc!")
    return "Error occured"

```

```

  newUpdate = {"$set" : update}

  #update execution with query and modification
  myUpdateResult = update_document(myQuery, newUpdate)

  #if specific query exists
  if (collection.find_one(myQuery) and myUpdateResult.modified_count == 1):
    #Print raw result info (useful!) & update results
    print(dumps(myUpdateResult.raw_result))
    print("Document updated!")
  elif (collection.find_one(myQuery) and myUpdateResult.modified_count == 0):
    print(dumps(myUpdateResult.raw_result))
    print("File has already been modified.")
  else:
    #return error message
    print("Document not found.")

modify_doc()
```

The prompt's requests and entries result in a "Document updated!" result as well as showing the MongoDB result: "nModified 1". If this said zero, the document wouldn't have been changed.

```
codio@opinion-trinity:~/workspace$ python stocks_update.py
Please enter document to be updated in the form: {"key" : "value"}
{"Ticker" : "JK"}
Please enter data to be updated in the form: {"key" : "value"}
{"Volume" : 47777}
{"updatedExisting": true, "nModified": 1, "ok": 1, "n": 1}
Document updated!
codio@opinion-trinity:~/workspace$
```



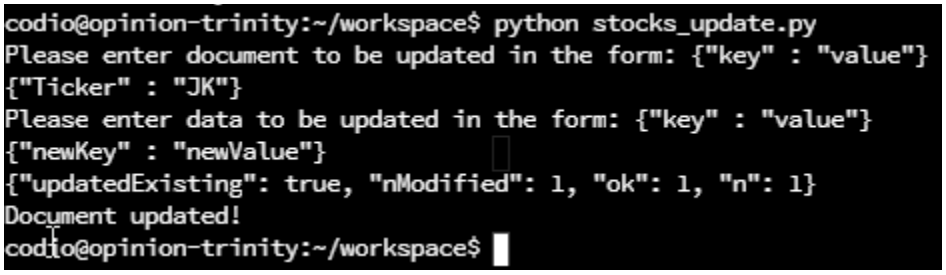


The database shows that the modification was successful.

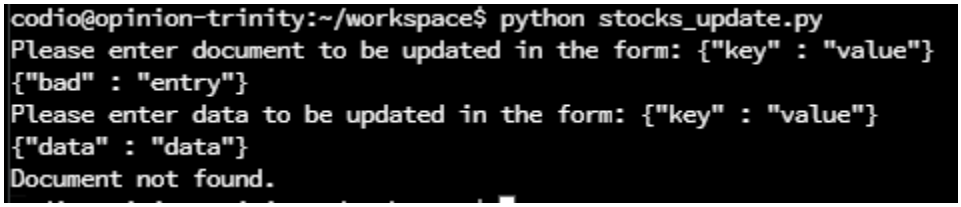
Additionally, this program will notice if the file was already modified and inform the user of that. (note that 'nModified' is now zero).

```
codio@opinion-trinity:~/workspace$ python stocks_update.py
Please enter document to be updated in the form: {"key" : "value"}
{"Ticker" : "JK"}
Please enter data to be updated in the form: {"key" : "value"}
{"Volume" : 47777}
{"updatedExisting": true, "nModified": 1, "ok": 1, "n": 1}
Document updated!
codio@opinion-trinity:~/workspace$ python stocks_update.py
Please enter document to be updated in the form: {"key" : "value"}
{"Ticker" : "JK"}
Please enter data to be updated in the form: {"key" : "value"}
{"Volume" : 47777}
{"updatedExisting": true, "nModified": 0, "ok": 1, "n": 1}
File has already been modified.
codio@opinion-trinity:~/workspace$
```

This program has two more abilities, it can insert new data into the file if the key is a new key:



The program can also inform the user that the file they wish to update could not be found.



This is the deletion program which deletes user defined files by requesting the ticker the user would like to delete, to be entered in a specific format, (as always).

```
import json
import bottle
from bson import json_util
from bson.json_util import dumps
from pymongo import MongoClient
from pymongo import errors

connection = MongoClient('localhost', 27017)
db = connection['market']
collection = db['stocks']

def delete_document(document):
  try:
    myDeleteResult = collection.delete_one(document)
    return myDeleteResult
  except errors.PyMongoError as pm:
    print("MongoDB returned error message", pm)
    abort(400, str(pm))
  return

def delete_doc():

  #request formatted data for deletion
  print('Please enter data/document to be deleted in the form: {"key" : "value"}')

  #take variable for deletion, query it
  try:
    myQuery = json.loads(raw_input())
  #return error if badly formatted data
  except ValueError:
    print("ValueError: wrongly formatted doc!")
    return "Error occured"

  #Deletion execution with query
  myDeleteResult = delete_document(myQuery)

  #if delete count isnt 1
  if (myDeleteResult.deleted_count != 1):
    #print error message
    print(dumps(myDeleteResult.raw_result))
    print("Document not found.")
  else:
    #return confirmation results
    print(dumps(myDeleteResult.raw_result))
    print("Document removed!")

delete_doc()
```

If the program is successfully deleted, the user will be informed. They will also be informed if the requested ‘ticker’ wasn’t found.

```
codio@opinion-trinity:~/workspace$ python stocks_delete.py
Please enter data/document to be deleted in the form: {"key" : "value"}
{"Ticker" : "JK"}
{"ok": 1, "n": 1}
Document removed!
codio@opinion-trinity:~/workspace$ python stocks_delete.py
Please enter data/document to be deleted in the form: {"key" : "value"}
{"Ticker" : "JK"}
{"ok": 1, "n": 0}
Document not found.
codio@opinion-trinity:~/workspace$
```

The user will also be told if their formatting was bad.

```
codio@opinion-trinity:~/workspace$ python stocks_delete.py
Please enter data/document to be deleted in the form: {"key" : "value"}
blegh!
ValueError: wrongly formatted doc!
codio@opinion-trinity:~/workspace$
```

Document Retrieval

The following program is a good example of a program that did not require an index, because it needed to scan the entire collection to capably carry out its work. This program reads from files and presents data to the user. Specifically, it takes a high and low user-defined number and returns to the user a count of all “50-Day Simple Moving Average” results that are between the high and low number.

```
codio@opinion-trinity:~/workspace$ python stocks_read1.py
Enter high integer for 50-Day Simple Moving Average
-0.5
Enter low integer for 50-Day Simple Moving Average
-0.7
11
codio@opinion-trinity:~/workspace$
```

```python
import json
from bson import json_util
from pymongo import MongoClient
from pymongo import errors
from bson.json_util import dumps, loads

connection = MongoClient('localhost', 27017)
db = connection['market']
collection = db['stocks']

#read query and print results
def moving_average(document):
  try:
    myReadResult = collection.find(document)
    #if specific query exists
    if (myReadResult != None):
      #convert to json and print
      print(dumps(myReadResult.count()))
    return
  except errors.PyMongoError as pm:
    print("MongoDB returned error message", pm)
    abort(400, str(pm))
    return

def read_main():

  print('Enter high integer for 50-Day Simple Moving Average')
  #store high value from user
  try:
    high = json.loads(raw_input())
  #return error if badly formatted data
  except ValueError:
    print("ValueError: wrongly formatted doc!")
    return "Error occured"

  print('Enter low integer for 50-Day Simple Moving Average')
  #store low value from user
  try:
    low = json.loads(raw_input())
  #return error if badly formatted data
  except ValueError:
    print("ValueError: wrongly formatted doc!")
    return "Error occured"

  #take document key/values for query
  myQuery = {"50-Day Simple Moving Average" : {"$lt" : high, "$gt" : low}}

  #send query to read function
  myReadResult = moving_average(myQuery)

read_main()
```

Additionally, the program informs the user if their entry was wrongly formatted.

```
codio@opinion-trinity:~/workspace$ python stocks_read1.py
Enter high integer for 50-Day Simple Moving Average
badEntry
ValueError: wrongly formatted doc!
```

This program will take a user defined string, such as “Medical Laboratories & Research” and search all industries for this string, returning a list of ‘ticker’ symbols for matching industries.

```
import json
from bson import json_util
from pymongo import MongoClient
from pymongo import errors
from bson.json_util import dumps

connection = MongoClient('localhost', 27017)
db = connection['market']
collection = db['stocks']

#query funtion
def find_industry(filt, proj):
  try:
    myReadResult = collection.find(filt, proj)
    #if specific query exists
    if (myReadResult.count() >= 1):
      #convert to json and print
      print(dumps(myReadResult))
    #if result found 0 files matching criteria
    elif (myReadResult.count() == 0):
      #return error message
      print("No Files Found For Industry:")
      print(dumps(filt))
    return
  except errors.PyMongoError as pm:
    print("MongoDB returned error message", pm)
    abort(400, str(pm))
    return


def read_main():

  print('Enter industry surrounded by double quotes')
  #store user string
  try:
    industry = json.loads(raw_input())
  #return error if badly formatted data
  except ValueError:
    print("ValueError: wrongly formatted doc!")
    return "Error occured"

  #reay query and search criteria and send to query funtion
  filterQ = {"Industry" : industry}
  projectionQ = {"Ticker" : 1, "_id" : 0}
  myReadResult = find_industry(filterQ, projectionQ)

read_main()
```

This program uses the compound index for industries and tickers.

```
codio@opinion-trinity:~/workspace$ python stocks_read2.py
Enter Industry surrounded by double quotes
"Medical Laboratories & Research"
[{"Ticker": "A"}, {"Ticker": "AIQ"}, {"Ticker": "ALR"}, {"Ticker": "BGMD"}, {"Ticker": "BRLI
"}, {"Ticker": "CBMX"}, {"Ticker": "CO"}, {"Ticker": "CVD"}, {"Ticker": "DGX"}, {"Ticker": "
ENZ"}, {"Ticker": "FMI"}, {"Ticker": "GHDX"}, {"Ticker": "HSKA"}, {"Ticker": "ICLR"}, {"Tick
er": "IRWD"}, {"Ticker": "LH"}, {"Ticker": "LIFE"}, {"Ticker": "LPDX"}, {"Ticker": "MTD"}, {
"Ticker": "NDZ"}, {"Ticker": "NEO"}, {"Ticker": "NVDQ"}, {"Ticker": "ONVO"}, {"Ticker": "PKI
"}, {"Ticker": "PMD"}, {"Ticker": "PRXL"}, {"Ticker": "Q"}, {"Ticker": "RDNT"}, {"Ticker": "
RGDX"}, {"Ticker": "ROSG"}, {"Ticker": "SPEX"}, {"Ticker": "TEAR"}, {"Ticker": "TMO"}, {"Tic
ker": "WAT"}, {"Ticker": "WX"}]
codio@opinion-trinity:~/workspace$
```

The program also informs the user if their entry was wrongly formatted:

```
codio@opinion-trinity:~/workspace$ python stocks_read2.py
Enter Industry surrounded by double quotes
no!
ValueError: wrongly formatted doc!
codio@opinion-trinity:~/workspace$
```

Additionally, the program will inform the user if no result matching the criteria was found:

```
codio@opinion-trinity:~/workspace$ python stocks_read2.py
Enter industry surrounded by double quotes
"Kitty Mittens"
No Files Found For Industry:
{"Industry": "Kitty Mittens"}
codio@opinion-trinity:~/workspace$
```

Finally, the last query is a bit more complex than the last two, as it uses an aggregated pipeline which typically modifies or adds to the query before performing the next command. For example, this aggregation first matches the first criteria, ‘sector’ which is provided by the user. Then it groups the results by ‘Industry’ with a summation of the industries ‘outstanding shares’ value.

```
import json
from bson import json_util
from pymongo import MongoClient
from pymongo import errors
from bson.json_util import dumps

connection = MongoClient('localhost', 27017)
db = connection['market']
collection = db['stocks']


def aggregate(filt, proj):
  try:
    myReadResult = collection.aggregate([filt, proj])
    #if specific query exists
    if (myReadResult != None):
      #convert to json and print
      print("Total oustanding shares grouped by industry:")
      print(dumps(myReadResult))
    return
  except errors.PyMongoError as pm:
    print("MongoDB returned error message", pm)
    abort(400, str(pm))
    return


def read_main():

  print('Enter sector surrounded by double quotes')
  #store user sector
  try:
    sector = json.loads(raw_input())
  #return error if badly formatted data
  except ValueError:
    print("ValueError: wrongly formatted doc!")
    return "Error occured"

  #store aggregation query
  filterQ = {"$match" : {"Sector" : sector}}
  projectionQ = {"$group" : {"_id" : "$Industry", "Total Outstanding Shares" : {
                    "$sum" : "$Shares Outstanding"}}}

  #if sector query doesnt exist & print error
  match = ({"Sector" : sector})
  if (collection.find_one(match) == None):
    print("No Matches Found For:")
    print(dumps(match))
  #send query aggregation to aggregation function
  else:
    myReadResult = aggregate(filterQ, projectionQ)

read_main()
```

The results of said aggregation. Note that the code needs to be more beautified for easier readability. I felt this was a job for version 2.0 as functionality comes first- then beautification.

```
codio@opinion-trinity:~/workspace$ python stocks_read3.py
Enter sector surrounded by double quotes
"Healthcare"
Total oustanding shares grouped by industry:
[{"_id": "Medical Practitioners", "Total Outstanding Shares": 19.24}, {"_id": "Medical I
nstruments & Supplies", "Total Outstanding Shares": 3512.9199999999983}, {"_id": "Drug M
anufacturers - Other", "Total Outstanding Shares": 3792.9299999999994}, {"_id": "Health
Care Plans", "Total Outstanding Shares": 3280.2200000000003}, {"_id": "Home Health Care"
, "Total Outstanding Shares": 193.44}, {"_id": "Specialized Health Services", "Total Out
standing Shares": 1923.1}, {"_id": "Biotechnology", "Total Outstanding Shares": 13893.71
9999999994}, {"_id": "Hospitals", "Total Outstanding Shares": 1246.4600000000003}, {"_id
": "Drug Delivery", "Total Outstanding Shares": 1730.4500000000003}, {"_id": "Medical Ap
pliances & Equipment", "Total Outstanding Shares": 8336.599999999999}, {"_id": "Drugs -
Generic", "Total Outstanding Shares": 1608.1800000000003}, {"_id": "Long-Term Care Facil
ities", "Total Outstanding Shares": 524.15}, {"_id": "Drug Manufacturers - Major", "Tota
l Outstanding Shares": 26805.450000000004}, {"_id": "Diagnostic Substances", "Total Outs
tanding Shares": 506.85}, {"_id": "Drug Related Products", "Total Outstanding Shares": 3
09.06}, {"_id": "Medical Laboratories & Research", "Total Outstanding Shares": 2495.2200
000000003}]
codio@opinion-trinity:~/workspace$ python stocks_read3.py
Enter sector surrounded by double quotes
"Basic Materials"
Total oustanding shares grouped by industry:
[{"_id": "Copper", "Total Outstanding Shares": 2057.18}, {"_id": "Oil & Gas Pipelines",
"Total Outstanding Shares": 9837.399999999998}, {"_id": "Synthetics", "Total Outstanding
 Shares": 323.25}, {"_id": "Independent Oil & Gas", "Total Outstanding Shares": 16417.19
0000000006}, {"_id": "Oil & Gas Equipment & Services", "Total Outstanding Shares": 7209.
119999999998}, {"_id": "Aluminum", "Total Outstanding Shares": 2464.68}, {"_id": "Chemic
als - Major Diversified", "Total Outstanding Shares": 5227.03}, {"_id": "Nonmetallic Min
eral Mining", "Total Outstanding Shares": 906.8599999999999}, {"_id": "Oil & Gas Drillin
g & Exploration", "Total Outstanding Shares": 23897.049999999996}, {"_id": "Major Integr
ated Oil & Gas", "Total Outstanding Shares": 28000.93}, {"_id": "Agricultural Chemicals"
, "Total Outstanding Shares": 2890.41}, {"_id": "Oil & Gas Refining & Marketing", "Total
 Outstanding Shares": 4408.25}, {"_id": "Silver", "Total Outstanding Shares": 1730.38999
99999999}, {"_id": "Industrial Metals & Minerals", "Total Outstanding Shares": 21226.47}
, {"_id": "Specialty Chemicals", "Total Outstanding Shares": 3442.2300000000005}, {"_id"
: "Gold", "Total Outstanding Shares": 12628.110000000004}, {"_id": "Steel & Iron", "Tota
l Outstanding Shares": 10221.039999999999}]
codio@opinion-trinity:~/workspace$
```

This program also considers if there is no such industry for the users input, and whether the input was wrongly formatted when it was entered:

```
codio@opinion-trinity:~/workspace$ python stocks_read3.py
Enter sector surrounded by double quotes
"Kitty Mittens"
No Matches Found For:
{"Sector": "Kitty Mittens"}
codio@opinion-trinity:~/workspace$
```

```
codio@opinion-trinity:~/workspace$ python stocks_read3.py
Enter sector surrounded by double quotes
no!
ValueError: wrongly formatted doc!
codio@opinion-trinity:~/workspace$
```

Now we come to making a REST API. First we develop a web-based service application in order to implement a RESTful application programming interface for the MongoDB database. I've named the API 'myrestapi' and given it a place in the home directory.

```
codio@opinion-trinity:~/workspace$ cd ..
codio@opinion-trinity:~$ ls
bottle-rest-service-master  log  starter-node-angular  startMongod.sh  workspace
codio@opinion-trinity:~$ mkdir myrestapi
codio@opinion-trinity:~$ ls
bottle-rest-service-master  log  myrestapi  starter-node-angular  startMongod.sh  workspace
codio@opinion-trinity:~$ cd /myrestapi/
-bash: cd: /myrestapi/: No such file or directory
codio@opinion-trinity:~$ cd myrestapi
codio@opinion-trinity:~/myrestapi$
```

Next, I switch to the directory and I import a 'rest_server' file to test out the API's functionality.

```
codio@opinion-trinity:~/myrestapi$ cp ~/workspace/rest_server.py ~/myrestapi/
codio@opinion-trinity:~/myrestapi$ ls
rest_server.py
codio@opinion-trinity:~/myrestapi$ python rest_server.py
Bottle v0.12.0 server starting up (using WSGIRefServer())...
Listening on http://localhost:8080/
Hit Ctrl-C to quit.
```

I noticed the ‘rest_server’ file wasn’t an executable so I went ahead and gave it some permissions (turning it green where before it was white) This probably doesn’t matter but I learned from using Linux that it’s a good practice.

```
codio@opinion-trinity:~/workspace$ chmod u+x ./APIread.py
codio@opinion-trinity:~/workspace$ chmod u+x ./APIupdate.py
codio@opinion-trinity:~/workspace$ ls
APIcreate.py  create.py  delete.py  read.py          stocks_delete.py  stocks_read3.py   time.py
APIread.py    curls.py   hello.py   rest_server.py   stocks_read1.py   stocks_update.py  update.py
APIupdate.py  datasets   read2.py   stocks_create.py  stocks_read2.py   test.py
codio@opinion-trinity:~/workspace$
```

After assuring my programs are executable, I test out the rest_server file and get a positive server response after using the CURLS (client side URL’s) on the client side:

```
codio@opinion-trinity:~/workspace$ curl http://localhost:8080/greeting
{ "id": 1, "content": "Hello, World!"}codio@opinion-trinity:~/workspace$
codio@opinion-trinity:~/workspace$ curl http://localhost:8080/greeting?name="Robert"
{ "id": 2, "content": "Hello, "Robert"}codio@opinion-trinity:~/workspace$ curl http://localhost:8080/greet
ing?name="Robert"[B
codio@opinion-trinity:~/workspace$ curl http://localhost:8080/greeting?name="Jessica"
{ "id": 3, "content": "Hello, "Jessica"}codio@opinion-trinity:~/workspace$
```

Server says:

```
codio@opinion-trinity:~/myrestapi$ python rest_server.py
Bottle v0.12.0 server starting up (using WSGIRefServer())...
Listening on http://localhost:8080/
Hit Ctrl-C to quit.

127.0.0.1 - - [15/Oct/2020 15:06:17] "GET /greeting HTTP/1.1" 200 38
127.0.0.1 - - [15/Oct/2020 15:06:47] "GET /greeting?name=Robert HTTP/1.1" 200 39
127.0.0.1 - - [15/Oct/2020 15:07:42] "GET /greeting?name=Jessica HTTP/1.1" 200 40
127.0.0.1 - - [15/Oct/2020 15:07:52] "GET /greeting HTTP/1.1" 200 38
```

This time the curls provide the user input. As you can see, the curl in the client above says within it “name=Robert” or “name=Jessica” . The program is designed to output a ‘return’ to the client with some “Hello world” code if the greeting doesn’t have a request for the name, or print out “Hello <name>” if there is a request for the name. We extract these names using our program which is designed to extract the request and print it in a preformatted string, otherwise the program aborts.

```
#!/usr/bin/pythonjavascript:void(0)
import json
from bson import json_util
import bottle
from bottle import route, run, request, abort, get

id = 0
#set up URI paths for REST service
@get('/greeting', method='GET')
def get_greeting():
    global id
    id = id + 1
    try:
        request.query.name
        name=request.query.name
        if name:
              string="{ \"id\": "+str(id)+", \"content\": \"Hello, \""+request.query.name+"\"}"
        else:
              string="{ \"id\": "+str(id)+", \"content\": \"Hello, World!\"}"
    except NameError:
        abort(404, 'No parameter for id %s' % id)

    if not string:
        abort(404, 'No id %s' % id)
    return json.loads(json.dumps(string, indent=4, default=json_util.default))


if __name__ == '__main__':
    #app.run(debug=True)
    run(host='localhost', port=8080)
```

Next I enable some CRUD functionality, starting with file creation. Here I design a program that creates new documents for a ticker symbol using data provided by a curl request with a POST http action.

```
import json
from bson import json_util
from pymongo import MongoClient
from bson.json_util import dumps
from pymongo import errors
from bottle import post, run, request, route

connection = MongoClient('localhost', 27017)
db = connection['market']
collection = db['stocks']

#insert execution
def insert_document(document):
  try:
    myInsertResult = collection.insert_one(document)
    print("Document created!")
    return True
  except errors.DuplicateKeyError as e:
    print("Duplicated key error")
    return False
  except errors.WriteError as we:
    print("MongoDB returned error message")
    return False
  except errors.WriteConcernError as wce:
    print("MongoDB returned error message")
    return False
```

```
27
28      #URI paths for REST service
29      @post('/stocks/api/v1.0/createStock/')
30      def main_create():
31          #create document by requests
32          myDocument = dict()
33          myDocument["Ticker"] = request.json["Ticker"]
34          myDocument["Profit Margin"] = request.json["Profit Margin"]
```

… (it's a long document, 66 lines of key/values)

```
83          myDocument["Average Volume"] = request.json["Average Volume"]
84          myDocument["EPS growth this year"] = request.json["EPS growth this year"]
85          myDocument["50-Day Simple Moving Average"] = request.json["50-Day Simple Moving Average"]
86
87          #pass tp insert function to insert document
88          myInsertResult = insert_document(myDocument)
89
90
91      if __name__ == '__main__':
92          run(host='localhost', port=8080)
93
94      main_create()
```

Remember when tickers are duplicate inserts an error is returned.

```
50-Day Simple Moving Average" : -0.589}
Duplicated key error
False
codio@opinion-trinity:~/workspace$
```

The document is created using a CURL provided by the client which contains a stream of information that the client entered. We enter this URL into the client side and the file will be created or rejected. Again, later this clunky feedback can be beautified for legibility.

```
codio@opinion-trinity:~/workspace$ curl -H "Content-Type: application/json" -X POST -d '{"Ticker" : "JK",
"Profit Margin" : -0.075,"Institutional Ownership" : 0.003,"EPS growth past 5 years" : 0.242,"Total Debt/
Equity" : 7.46,"Current Ratio" : 0.7,"Return on Assets" : -0.074,"Sector" : "Services","P/S" : 0.06,"Chan
ge from Open" : 0.0205,"Performance (YTD)" : -0.9353,"Performance (Week)" : -0.137,"Quick Ratio" : 0.7,"P
/B" : 0.82,"EPS growth quarter over quarter" : 0.119,"Performance (Quarter)" : -0.767,"200-Day Simple Mov
ing Average" : -0.7382,"Shares Outstanding" : 0.99,"52-Week High" : -0.9308,"P/Cash" : 1,"Change" : 0.068
7,"Analyst Recom" : 3,"Volatility (Week)" : 0.0996,"Country" : "USA","Return on Equity" : 6.889,"50-Day L
ow" : 0.1318,"Price" : 2.49,"50-Day High" : -0.7658,"Return on Investment" : -0.011,"Shares Float" : 3.09
,"Industry" : "Management Services","Beta" : -2.5,"Sales growth quarter over quarter" : 9.286,"Operating
Margin" : -0.017,"EPS (ttm)" : -5.97,"52-Week Low" : 0.1318,"Average True Range" : 0.37,"Company" : "Jess
icus Kilbourneous Inc.","Gap" : 0.0472,"Relative Volume" : 3.24,"Volatility (Month)" : 0.0714,"Market Cap
" : 2.3,"Volume" : 46666,"Gross Margin" : 0.292,"Performance (Half Year)" : -0.7376,"Relative Strength In
dex (14)" : 22.46,"Insider Ownership" : 0.071,"20-Day Simple Moving Average" : -0.4444,"Performance (Mont
h)" : -0.6331,"LT Debt/Equity" : 4.11,"Average Volume" : 15.83,"EPS growth this year" : 0.791,"50-Day Sim
ple Moving Average" : -0.589}' http://localhost:8080/stocks/api/v1.0/createStock/
codio@opinion-trinity:~/workspace$
```

When the document is successfully created, we get a “Document created! True” reply on the server side and a return code of ‘200’ meaning that the creating was successful.

```
codio@opinion-trinity:~/myrestapi$ python APIcreate.py
Bottle v0.12.0 server starting up (using WSGIRefServer())...
Listening on http://localhost:8080/
Hit Ctrl-C to quit.

Document created!
127.0.0.1 - - [19/Oct/2020 19:07:59] "POST /stocks/api/v1.0/createStock/ HTTP/1.1" 200 0
```

Otherwise we get a duplicate error or write concern error.

The next program allows the user to update the value in the ‘volume’ key for document of their choosing. First it takes from the curl the users document they would like to modify, and then it gets the value for ‘volume’ provided by the curl.

```
codio@opinion-trinity:~/workspace$ curl -H "Content-Type: application/json" -X PUT -d '{"Volume" : 47474
}' http://localhost:8080/stocks/api/v1.0/updateStock?ticker="JK"
codio@opinion-trinity:~/workspace$ curl -H "Content-Type: application/json" -X PUT -d '{"Volume" : 47474
}' http://localhost:8080/stocks/api/v1.0/updateStock?ticker="JK"
codio@opinion-trinity:~/workspace$
```

```python
import json
from bson import json_util
from bson.json_util import dumps
from pymongo import MongoClient
from pymongo import errors
from bottle import get, put, route, run, request, abort

connection = MongoClient('localhost', 27017)
db = connection['myDB']
collection = db['myCollection']

#update funtion
def update_document(query, newMod):
  try:
    myUpdateResult = collection.update_one(query, newMod)
    return myUpdateResult
  except errors.PyMongoError as pm:
    print("MongoDB returned error message", pm)
    abort(400, str(pm))
    return

#URI paths for REST service
@put('/stocks/api/v1.0/updateStock')
def main_update():
  ticker = request.params.get('ticker')
  myQuery = {"Ticker" : ticker}
  modify = request.json["Volume"]
  newUpdate = {"$set" : {"Volume" : modify}}

  #send query and update to update function
  myUpdateResult = update_document(myQuery, newUpdate)
```

```
  #if query exists & was modified
  if (collection.find_one(myQuery) and myUpdateResult.modified_count == 1):
    #print raw info & update confirmation
    print(dumps(myUpdateResult.raw_result))
    print("Document updated!")
  #if query exists & was not modified
  elif (collection.find_one(myQuery) and myUpdateResult.modified_count == 0):
    #print raw info & info message
    print(dumps(myUpdateResult.raw_result))
    print("File has already been modified.")
  else:
    #print raw info & return error message
    print(dumps(myUpdateResult.raw_result))
    print("Document not found.")

if __name__ == '__main__':
    run(host='localhost', port=8080, debug=True)

main_update()
```

If the query exists and was modified, we get a “Document Updated!” reply. If the query exist and the document was not modified, we get a “File has already been modified” reply, otherwise the query couldn’t find the document. The program also has the ability to return MongoDB errors. Note that we could make ‘volume’ a user specified field and update anything we like.

```
codio@opinion-trinity:~/myrestapi$ python APIupdate.py
Bottle v0.12.0 server starting up (using WSGIRefServer())...
Listening on http://localhost:8080/
Hit Ctrl-C to quit.

{"updatedExisting": true, "nModified": 1, "ok": 1, "n": 1}
Document updated!
127.0.0.1 - - [17/Oct/2020 15:59:02] "PUT /stocks/api/v1.0/updateStock?ticker=JK HTTP/1.1" 200 0
{"updatedExisting": true, "nModified": 0, "ok": 1, "n": 1}
File has already been modified.
127.0.0.1 - - [17/Oct/2020 15:59:05] "PUT /stocks/api/v1.0/updateStock?ticker=JK HTTP/1.1" 200 0
```

```
"Volume" : 46666,
"Beta" : -2.5,
"P/Cash" : 1,
"Sales growth quarter over quarter" : 9.286,
"EPS growth this year" : 0.791,
"Relative Strength Index (14)" : 22.46,
"Ticker" : "JK",
```

Update before change. Notice ‘volume’.

```
"Volume" : 47474,
"Beta" : -2.5,
"P/Cash" : 1,
"Sales growth quarter over quarter" : 9.286,
"EPS growth this year" : 0.791,
"Relative Strength Index (14)" : 22.46,
"Ticker" : "JK",
```

Update after change. Note ‘volume’.

Next, we have delete functionality, that takes a user curl and extracts the ticker name for the file they want deleted. If the deletion is successful, the document is removed. Otherwise it's likely the document wasn't found because it doesn't exist, and we get an error message.

```
codio@opinion-trinity:~/workspace$ curl -H "Content-Type: application/json" -X DELETE -d '{"Ticker" : "JK"}
' http://localhost:8080/stocks/api/v1.0/deleteStock
codio@opinion-trinity:~/workspace$ curl -H "Content-Type: application/json" -X DELETE -d '{"Ticker" : "JK"}
' http://localhost:8080/stocks/api/v1.0/deleteStock
codio@opinion-trinity:~/workspace$
```

```python
import json
import bottle
from bson import json_util
from bson.json_util import dumps
from pymongo import MongoClient
from pymongo import errors
from bottle import delete, route, run, request, abort

connection = MongoClient('localhost', 27017)
db = connection['myDB']
collection = db['myCollection']

def delete_document(document):
  try:
    myDeleteResult = collection.delete_one(document)
    return myDeleteResult
  except errors.PyMongoError as pm:
    print("MongoDB returned error message", pm)
    abort(400, str(pm))
  return

@delete('/stocks/api/v1.0/deleteStock')
def delete_doc():
  #take variable for deletion, query it
  ticker = request.json["Ticker"]
  myQuery = {"Ticker" : ticker}

  #Deletion execution with query
  myDeleteResult = delete_document(myQuery)

  #if delete count isnt 1
  if (myDeleteResult.deleted_count != 1):
    #print error message
    print(dumps(myDeleteResult.raw_result))
    print("Document not found.")
  else:
    #return confirmation results
    print(dumps(myDeleteResult.raw_result))
    print("Document removed!")

if __name__ == '__main__':
    run(host='localhost', port=8080, debug=True)

delete_doc()
```

```
codio@opinion-trinity:~/myrestapi$ python APIdelete.py
Bottle v0.12.0 server starting up (using WSGIRefServer())...
Listening on http://localhost:8080/
Hit Ctrl-C to quit.

{"ok": 1, "n": 1}
Document removed!
127.0.0.1 - - [17/Oct/2020 16:50:27] "DELETE /stocks/api/v1.0/deleteStock HTTP/1.1" 200 0
{"ok": 1, "n": 0}
Document not found.
127.0.0.1 - - [17/Oct/2020 16:51:13] "DELETE /stocks/api/v1.0/deleteStock HTTP/1.1" 200 0
```

Next we have simple' read' functionality which will query the given file.. The user enters a ticker to read a file by the ticker name, and their results are returned.

```
codio@opinion-trinity:~/workspace$ curl -H "Content-Type: application/json" -X GET -d '{"Ticke
r" : "JK"}' http://localhost:8080/stocks/api/v1.0/getStock
codio@opinion-trinity:~/workspace$ curl -H "Content-Type: application/json" -X GET -d '{"Ticke
r" : "JK"}' http://localhost:8080/stocks/api/v1.0/getStock
codio@opinion-trinity:~/workspace$
```

Here the file is read, and then I delete the file, to show that next time we read we get an error that there was "No File Found With That Criteria".

```
codio@opinion-trinity:~/myrestapi$ python APIread.py
Bottle v0.12.0 server starting up (using WSGIRefServer())...
Listening on http://localhost:8080/
Hit Ctrl-C to quit.

[{"50-Day High": -0.7658, "Return on Equity": 6.889, "Current Ratio": 0.7, "20-Day Simple Movi
ng Average": -0.4444, "Relative Volume": 3.24, "Analyst Recom": 3, "Average Volume": 15.83, "P
/S": 0.06, "Performance (Half Year)": -0.7376, "52-Week Low": 0.1318, "Insider Ownership": 0.0
71, "Shares Float": 3.09, "50-Day Simple Moving Average": -0.589, "P/B": 0.82, "LT Debt/Equity
": 4.11, "50-Day Low": 0.1318, "Price": 2.49, "200-Day Simple Moving Average": -0.7382, "Gross
 Margin": 0.292, "Gap": 0.0472, "Volume": 46666, "Beta": -2.5, "Sales growth quarter over quar
ter": 9.286, "EPS growth this year": 0.791, "Relative Strength Index (14)": 22.46, "Ticker": "
JK", "Change": 0.0687, "Change from Open": 0.0205, "Performance (Quarter)": -0.767, "Instituti
onal Ownership": 0.003, "Country": "USA", "Industry": "Management Services", "Return on Assets
": -0.074, "Performance (YTD)": -0.9353, "52-Week High": -0.9308, "Volatility (Week)": 0.0996,
 "EPS (ttm)": -5.97, "EPS growth past 5 years": 0.242, "EPS growth quarter over quarter": 0.11
9, "Average True Range": 0.37, "Sector": "Services", "Company": "Jessicus Kilbourneous Inc.",
"Shares Outstanding": 0.99, "Quick Ratio": 0.7, "P/Cash": 1, "Return on Investment": -0.011, "
Performance (Week)": -0.137, "Volatility (Month)": 0.0714, "Total Debt/Equity": 7.46, "Perform
ance (Month)": -0.6331, "Profit Margin": -0.075, "Operating Margin": -0.017, "_id": {"$oid": "
5f8b3f0b158d440742ecdf5a"}, "Market Cap": 2.3}]
127.0.0.1 - - [17/Oct/2020 19:01:34] "GET /stocks/api/v1.0/getStock HTTP/1.1" 200 0
No File Found With That Criteria.
127.0.0.1 - - [17/Oct/2020 19:02:04] "GET /stocks/api/v1.0/getStock HTTP/1.1" 200 0
```

```
> db.myCollection.find({"Ticker" : "JK"}).length()
1
> db.myCollection.remove({"Ticker" : "JK"})
WriteResult({ "nRemoved" : 1 })
> db.myCollection.find({"Ticker" : "JK"}).length()
0
```

```python
import json
from bson import json_util
from pymongo import MongoClient
from pymongo import errors
from bson.json_util import dumps
from bottle import get, route, run, request

connection = MongoClient('localhost', 27017)
db = connection['myDB']
collection = db['myCollection']

#read function
def read_document(document):
  try:
    myReadResult = collection.find(document)
    #if specific query exists
    if (myReadResult.count() != 0):
      #convert to json and print
      print(dumps(myReadResult))
    #if result found 0 matching files
    elif (myReadResult.count() == 0):
      #return error message
      print("No File Found With That Criteria.")
    return
  except errors.PyMongoError as pm:
    print("MongoDB returned error message", pm)
    abort(400, str(pm))
    return

#URI paths for REST service
@get('/stocks/api/v1.0/getStock')
def main_read():
  #take and value for query
  ticker = request.json["Ticker"]
  myQuery = {"Ticker" : ticker}

  #send to read funtion
  myReadResult = read_document(myQuery)

if __name__ == '__main__':
    run(host='localhost', port=8080, debug=True)

main_read()

```

The limits of what's possible with the REST API and MongoDB are that of user imagination. I would highly recommend that your knowledgeable sock experts communicate their ideas about what they would like to see implemented into the system. For now, I have a few mock example queries they can study to get some ideas about what's possible.

First is a simple query that selects and presents specific stock summary information from a user-derived list of ticker symbols. In this query we create a stock report for the list of user ticker symbols using the data provided by the request in the curl.

```
codio@opinion-trinity:~/workspace$ curl -H "Content-Type: application/json" -X POST -d '{"array" : ["A
A", "BA", "T"]}' http://localhost:8080/stocks/api/v1.0/stockReport
codio@opinion-trinity:~/workspace$
```

Note that the in curl there is an array of tickers within it: {"array" : ["AA", "BA", "T"]}. Execution prints a big block of code that we can beautify into whatever look you may want.

```
codio@opinion-trinity:~/myrestapi$ python APIstockReport.py
Bottle v0.12.0 server starting up (using WSGIRefServer())...
Listening on http://localhost:8080/
Hit Ctrl-C to quit.

[{"Forward P/E": 21.35, "50-Day High": -0.0925, "Return on Equi
ansactions": 0.1031, "20-Day Simple Moving Average": -0.0192, "
```

```
 growth next 5 years": 0.0647, "Shares Outstanding": 5315, "Quick Ratio": 0.5, "Total Debt/
eturn on Investment": 0.062, "Volatility (Month)": 0.0138, "EPS growth quarter over quarter
Margin": 0.56, "Profit Margin": 0.058, "Relative Strength Index (14)": 49.51, "Performance
 "_id": {"$oid": "5285380dbb1177ca391c2ea0"}, "Company": "AT&T, Inc.", "Earnings Date": {"$
000}}]
127.0.0.1 - - [17/Oct/2020 21:57:18] "POST /stocks/api/v1.0/stockReport HTTP/1.1" 200 0
```

For now, what we care about is that the query was successful. And it was, which is indicated by the '200' code at the end.

```python
import json
from bson import json_util
from pymongo import MongoClient
from pymongo import errors
from bson.json_util import dumps
from bottle import post, route, run, request

connection = MongoClient('localhost', 27017)
db = connection['market']
collection = db['stocks']

#read function
def read_document(document):
  try:
    myReadResult = collection.find(document)
    #if query count isnt 0
    if (myReadResult.count() != 0):
      #convert to json and print
      print(dumps(myReadResult))
    #if result found 0 matching files
    elif (myReadResult.count() == 0):
      #return error message
      print("No Files Found With That Criteria.")
    return
  except errors.PyMongoError as pm:
    print("MongoDB returned error message", pm)
    abort(400, str(pm))
    return

#URI paths for REST service
@post('/stocks/api/v1.0/stockReport')
def main_read():
  #take value for query
  array = request.json["array"]
  myQuery = {"Ticker" : {"$in" : array}}

  #send to read funtion
  myReadResult = read_document(myQuery)

if __name__ == '__main__':
    run(host='localhost', port=8080, debug=True)

main_read()
```

Finally, we perform an aggregation, the most complex query of all, but also the most powerful. The aggregation program extracts a company name from a curl, and inserts it into an aggregation pipeline type of search. This aggregation searches the collection and reports a portfolio on the top five shares (given some criteria to look for), first using a user-defined industry to conduct the search. We match this industry (for which we have an index) and group

the top “max” companies by several factors. Researching a little on stocks I decided to go with ‘relative strength index’ and the highest 200-Day Simple Moving Average (SMA). Then these companies are sorted by their highest strength index. The ‘regex’ code looks within the given names for industries and conducts searches for single words. For example, here we search for “Telecom”, a word which can only be found embedded within a given industry’s name.

```
codio@opinion-trinity:~$ curl -H "Content-Type: application/json" -X GET -d '{"Industry" : "Telecom"}' http://localh
ost:8080/stocks/api/v1.0/industryReport
codio@opinion-trinity:~$
```

```python
import json
from bson import json_util
from pymongo import MongoClient
from pymongo import errors
from bson.json_util import dumps
from bottle import get, route, run, request, abort

connection = MongoClient('localhost', 27017)
db = connection['market']
collection = db['stocks']

#aggregate Function
def aggregateFn(aggreg):
  try:
    myReadResult = collection.aggregate(aggreg)
    print(myReadResult)
    #if aggregation does not equal 'None'
    if (myReadResult != None):
      #convert to json and print
      print("Top five shares grouped by company, strength, 200-Day SMA.")
      print(dumps(myReadResult))
    return
  except Exception as pm:
    print(dumps("MongoDB returned error message", pm))

#URI for REST service
@get('/stocks/api/v1.0/industryReport')
def read_main():

  #take value for query
  industry = request.json["Industry"]

  #aggregation filtering & projection criteria
  aggregationQ = [{"$match" : {"Industry" : {"$regex" : ".*"+industry+".*"}}}, {"$sort" : {"HighestStrength" : -1}},
                  {"$group" :  {"_id" : "$Company", "HighestStrength" :
                   {"$max" : "$Relative Strength Index (14)"},"Highest200-DaySMA" :
                   {"$max" : "$200-Day Simple Moving Average"}}},
                   {"$limit" : 5}]

  #if industry query doesnt exist, print error
  match = {"Industry" : {"$regex" : ".*"+industry+".*"}}
  if (collection.find(match) == None):
    print("No Matches Found For:")
    print(dumps(match))
  #else send variables to aggregation function
  else:
    myReadResult = aggregateFn(aggregationQ)

if __name__ == '__main__':
    run(host='localhost', port=8080, debug=True)

read_main()
```

```
codio@opinion-trinity:~/myrestapi$ python APIindustryReport.py
Bottle v0.12.0 server starting up (using WSGIRefServer())...
Listening on http://localhost:8080/
Hit Ctrl-C to quit.

<pymongo.command_cursor.CommandCursor object at 0x7f0037983d10>
Top five shares grouped by company, strength, 200-Day SMA.
[{"HighestStrength": 40.61, "_id": "Tata Communications Limited", "Highest200-DaySMA": -0.1634}, {"HighestStrength":
 35.17, "_id": "Philippine Long Distance Telephone Co.", "Highest200-DaySMA": -0.051}, {"HighestStrength": 43.81, "_
id": "Orange", "Highest200-DaySMA": 0.2023}, {"HighestStrength": 33.99, "_id": "KT Corp.", "Highest200-DaySMA": -0.0
637}, {"HighestStrength": 32.84, "_id": "iPass Inc.", "Highest200-DaySMA": -0.1234}]
127.0.0.1 - - [21/Oct/2020 00:06:09] "GET /stocks/api/v1.0/industryReport HTTP/1.1" 200 0
```

MongoDB will return an error if there was something wrong with the aggregation query.

```
codio@opinion-trinity:~/myrestapi$ python APIindustryReport.py
Bottle v0.12.0 server starting up (using WSGIRefServer())...
Listening on http://localhost:8080/
Hit Ctrl-C to quit.

<pymongo.command_cursor.CommandCursor object at 0x7f614a1f5a10>
Top five shares grouped by company, strength, 200-Day SMA.
[]
"MongoDB returned error message"
```

If no files where found that match the industry given, the program will return an error.

```
codio@opinion-trinity:~/myrestapi$ python APIindustryReport.py
Bottle v0.12.0 server starting up (using WSGIRefServer())...
Listening on http://localhost:8080/
Hit Ctrl-C to quit.

No Matches Found For:
{"Industry": {"$regex": ".*tetecom.*"}}
127.0.0.1 - - [22/Oct/2020 00:47:26] "GET /stocks/api/v1.0/industryReport HTTP/1.1" 200 0
```

Given the right feedback from some knowledgeable stock market peers, I'm sure these reports could increase in quality, quantity and pay for themselves ten times over- proving to be a great addition as well as an asset to your business.