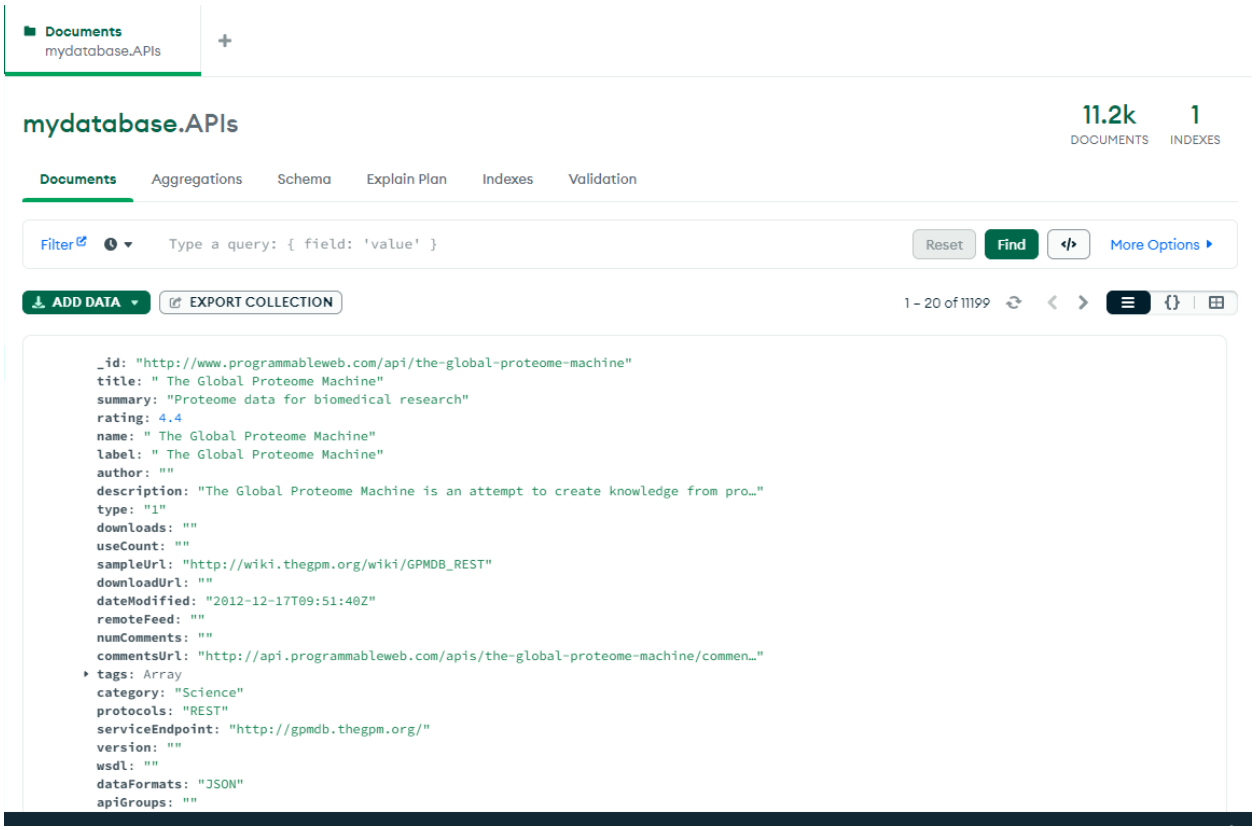
Web services and Service Oriented Architecture Programming Assignment 2

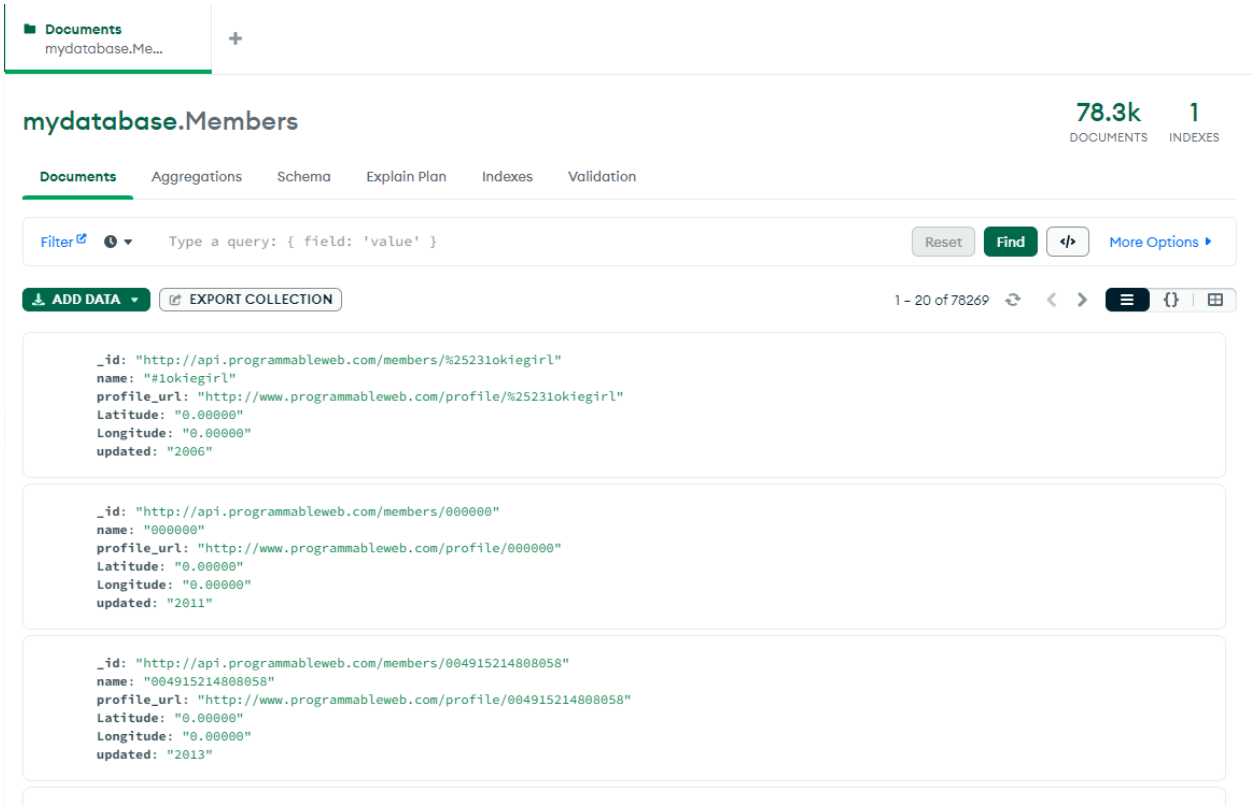
1. Design of the application:

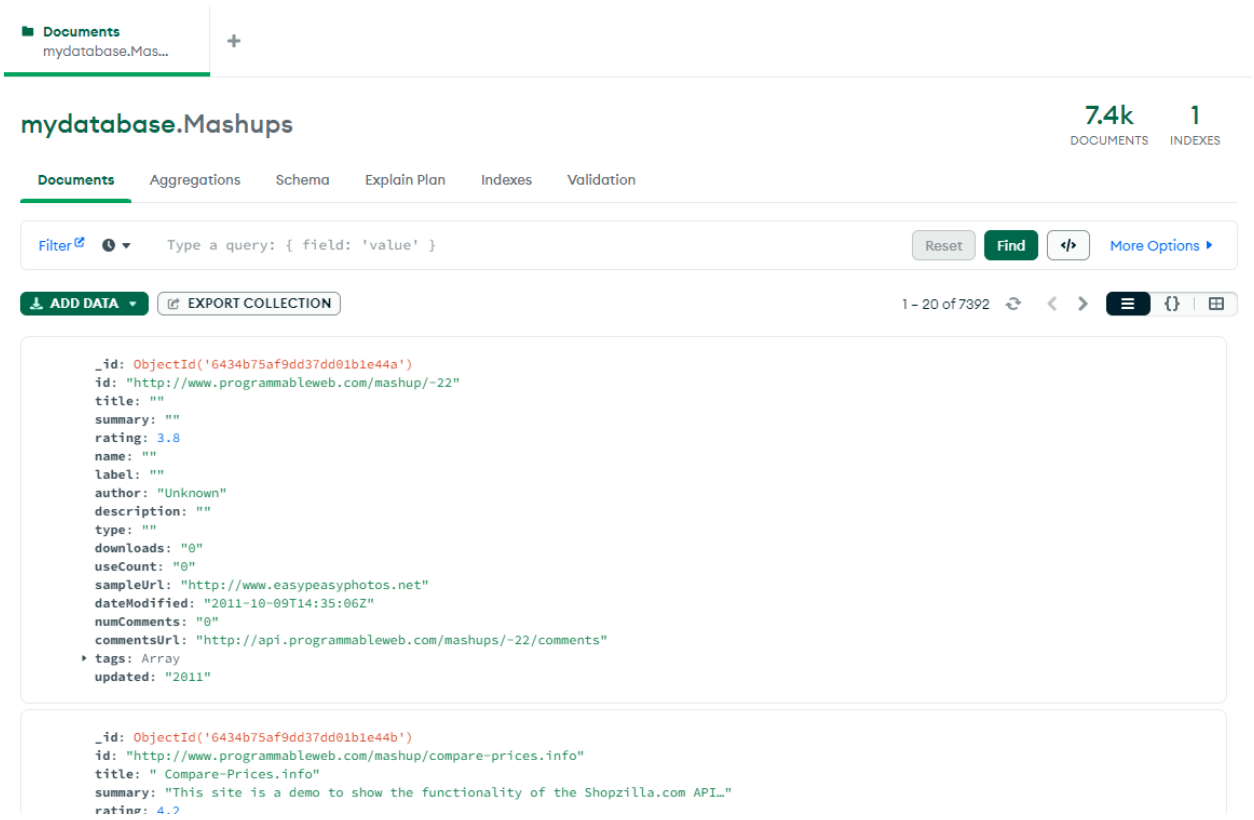
I have designed an application that parses data, stores it in mongoDB and analyze it. The design in simple where the app.py is the backend of the application which uses Flask micro-framework to process the data and sends it to mongoDB, once the data is stored in the database, there is a front-end aspect of the application which performs all the querying operations and displays the required data.

2. Design of database and implementation:

I have created three collections to store API, Mashup and Members data. I am using id of API as the primary ID, processing few data fields to make the querying easier like year, rating and so on. The year is updated to match a valid year, rating is converted to float and sent over to mongoDB.

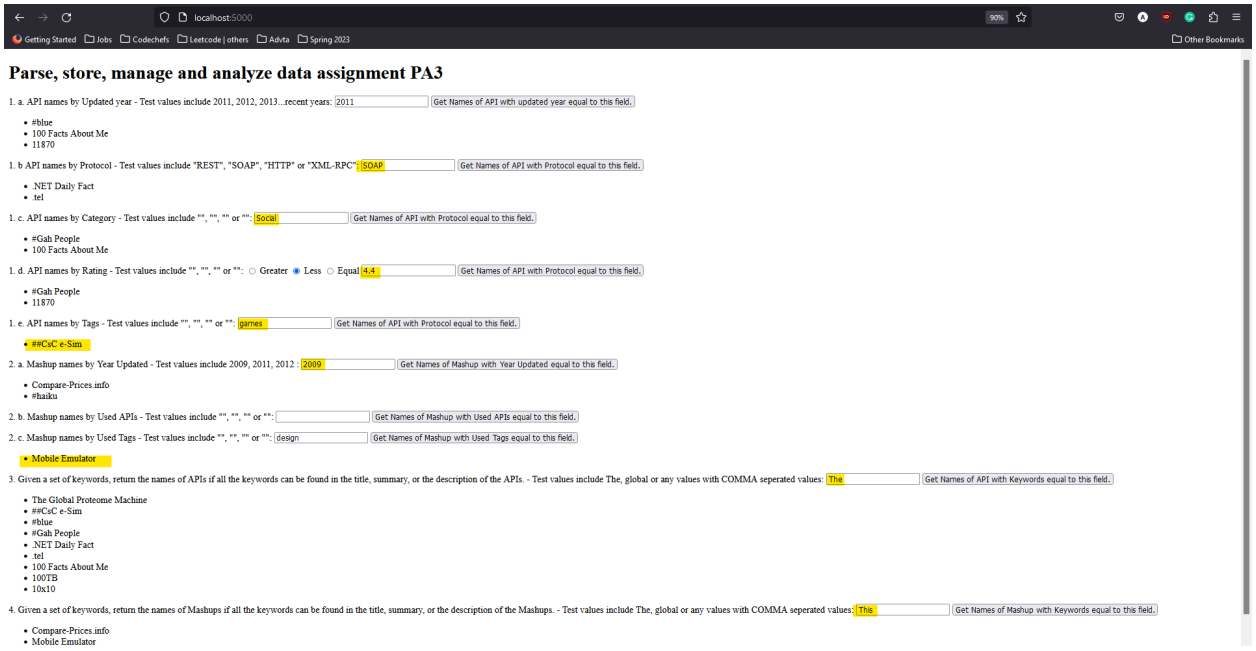

Documents
mydatabase.APIs
mydatabase.APIs
11.2k
DOCUMENTS
1
INDEXES
Documents
Aggregations
Schema
Explain Plan
Indexes
Validation
Filter
Type a query: { field: 'value' }
Reset
Find
More Options
ADD DATA
EXPORT COLLECTION
1 - 20 of 11199
_id: "http://www.programmableweb.com/api/the-global-proteome-machine"
title: " The Global Proteome Machine"
summary: "Proteome data for biomedical research"
rating: 4.4
name: " The Global Proteome Machine"
label: " The Global Proteome Machine"
author: ""
description: "The Global Proteome Machine is an attempt to create knowledge from pro…"
type: "1"
downloads: ""
useCount: ""
sampleUrl: "http://wiki.thegpm.org/wiki/GPMDB_REST"
downloadUrl: ""
dateModified: "2012-12-17T09:51:40Z"
remoteFeed: ""
numComments: ""
commentsUrl: "http://api.programmableweb.com/apis/the-global-proteome-machine/commen…"
tags: Array
category: "Science"
protocols: "REST"
serviceEndpoint: "http://gpmdb.thegpm.org/"
version: ""
wsdl: ""
dataFormats: "JSON"
apiGroups: ""


Documents
mydatabase.Me...
mydatabase.Members
78.3k
DOCUMENTS
1
INDEXES
Documents
Aggregations
Schema
Explain Plan
Indexes
Validation
Filter
Type a query: { field: 'value' }
Reset
Find
More Options
ADD DATA
EXPORT COLLECTION
1 - 20 of 78269
_id: "http://api.programmableweb.com/members/%25231okiegirl"
name: "#1okiegirl"
profile_url: "http://www.programmableweb.com/profile/%25231okiegirl"
Latitude: "0.00000"
Longitude: "0.00000"
updated: "2006"
_id: "http://api.programmableweb.com/members/000000"
name: "000000"
profile_url: "http://www.programmableweb.com/profile/000000"
Latitude: "0.00000"
Longitude: "0.00000"
updated: "2011"
_id: "http://api.programmableweb.com/members/004915214808058"
name: "004915214808058"
profile_url: "http://www.programmableweb.com/profile/004915214808058"
Latitude: "0.00000"
Longitude: "0.00000"
updated: "2013"



Once the data is stored in database, the user interacts with UI, from which query field is extracted and routed accordingly.

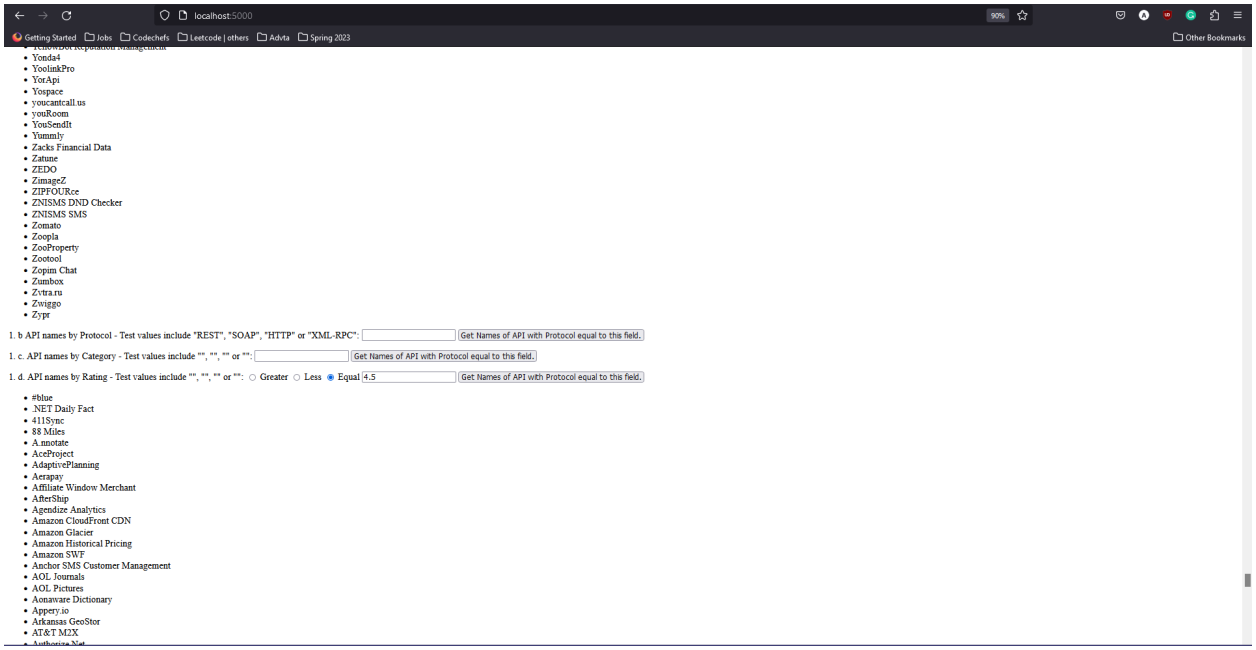
For ex, to get API names by updated year: we can enter the valid value in the UI and get back the names of the API from the server.

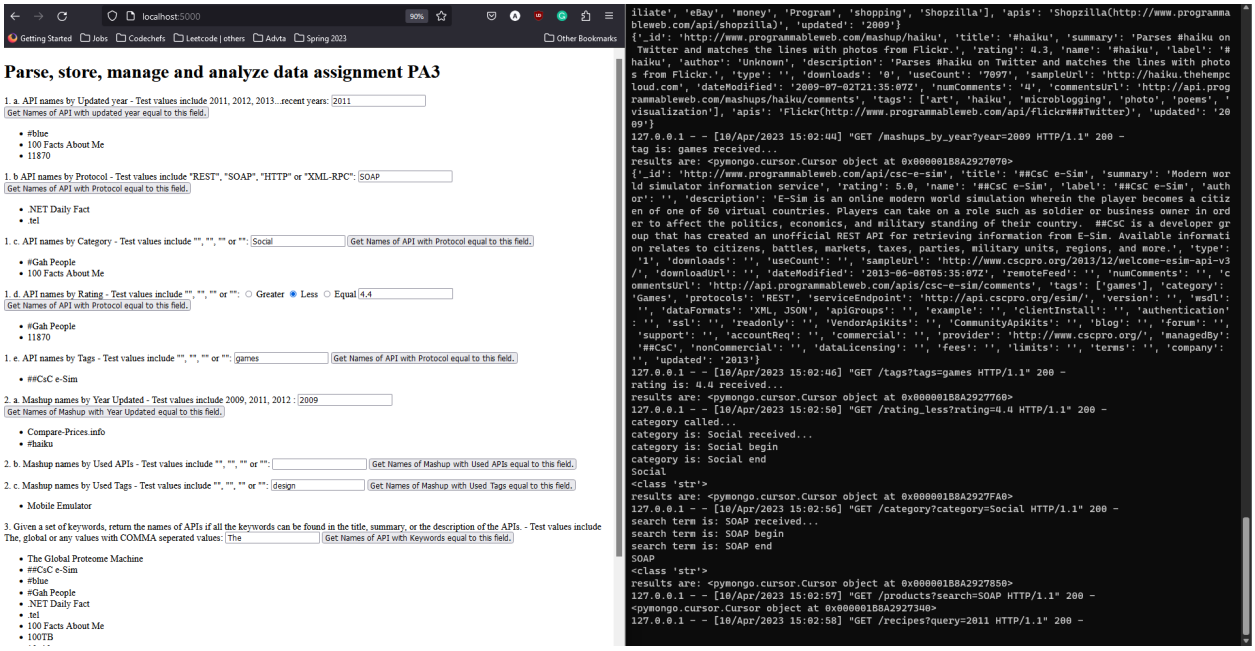


The above image shows all the queries working well.

The queries are invoked from the flask route, which interacts with the database

Technology used:
I have used Flask API to build the web service, MongoDB as the database.





3. README

Server side:

Pip3 install flask

```
from flask import Flask, request
from flask import render_template,redirect
from flask_pymongo import PyMongo
from flask import jsonify
from bson.json_util import dumps
```

Install mongoDB

Client side:

Once these requirements are installed on to the system, we can run the server by running the flask.
Flask run

Once we make sure the server is running, we can call just visit localhost:5000 to visit the home page.

We then call localhost:5000/api -> to parse and store data in the database.

After that, we can return to home page, localhost:5000 where we can query all the required data.