Apollo Hospital Database Management

NoSQL Assignment

-Pooja Deswal

Introduction:

The aim of the project is to build a REST API on Apollo Hospital database using python, flask and MongoDB. Apollo Hospitals Enterprise Limited is an Indian hospital chain based in Chennai, India. It was founded by Dr. Prathap C. Reddy in 1983 as the first corporate health care provider in India. Several of the Apollo's hospitals have been among the first in India to receive international healthcare accreditation by the America-based Joint Commission International (JCI) as well as 13 NABH National Accreditation Board for Hospitals & Healthcare Providers hospitals.

Software Used:

1. **MongoDB**
   MongoDB is a document-oriented NoSQL database used for high volume data storage. Instead of using tables and rows as in the traditional relational databases, MongoDB makes use of collections and documents. Documents consist of key-value pairs which are the basic unit of data in MongoDB. Collections contain sets of documents and function which is the equivalent of relational database tables. MongoDB is a database which came into light around the mid-2000s.

2. **Flask**
   Flask is a web application framework written in Python. Armin Ronacher, who leads an international group of Python enthusiasts named Pocco, develops it. Flask is based on Werkzeug WSGI toolkit and Jinja2 template engine. Both are Pocco projects.

3. **Python**
   Python is a high-level, general-purpose and a very popular programming language. Python programming language (latest Python 3) is being used in web development, Machine Learning applications, along with all cutting edge technology in Software Industry. Python Programming Language is very well suited for Beginners, also for experienced programmers with other programming languages like C++ and Java.

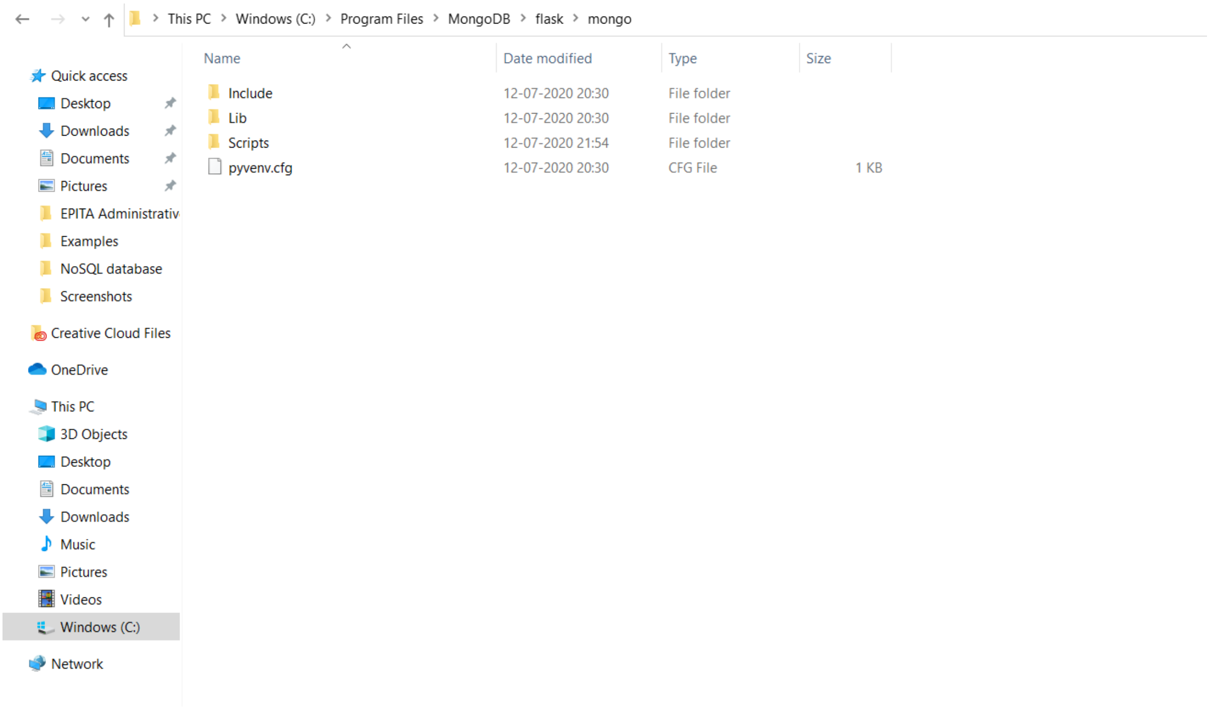
4. **Postman**
   Postman is a collaboration platform for API development. Postman's features simplify each step of building an API and streamline collaboration so you can create better APIs—faster.

Installation Guide:

1. Check if python is installed in system
   a. Open CMD as administrator
   b. Type *pip list* and see if python is listed
   c. If not, run command *pip install python*
2. In C: drive, create folder *data* and inside data another folder *db*
3. Go to Task Manager>Services>MongoDB and check if service is running
4. Go to MongoDB.com and download community server. An .msi installer file will be downloaded.
5. Once file is download, Install MongoDB by running .msi file. Follow steps and do complete installation.
6. Once it is installed, MongoDB compass will be installed. You will be able to connect it to the database on *localhost* and port *27017*.
7. By default, three databases will be created.

Creating the database

1. Go to CMD and run pip list to check tools installed.
2. Create a new folder flask anywhere in the system. Inside flask folder, create another folder named mongo
3. In CMD, go to Mongo directory, create a virtual environment, using command *py -m venv .*
4. Below folders will be created



This PC > Windows (C:) > Program Files > MongoDB > flask > mongo

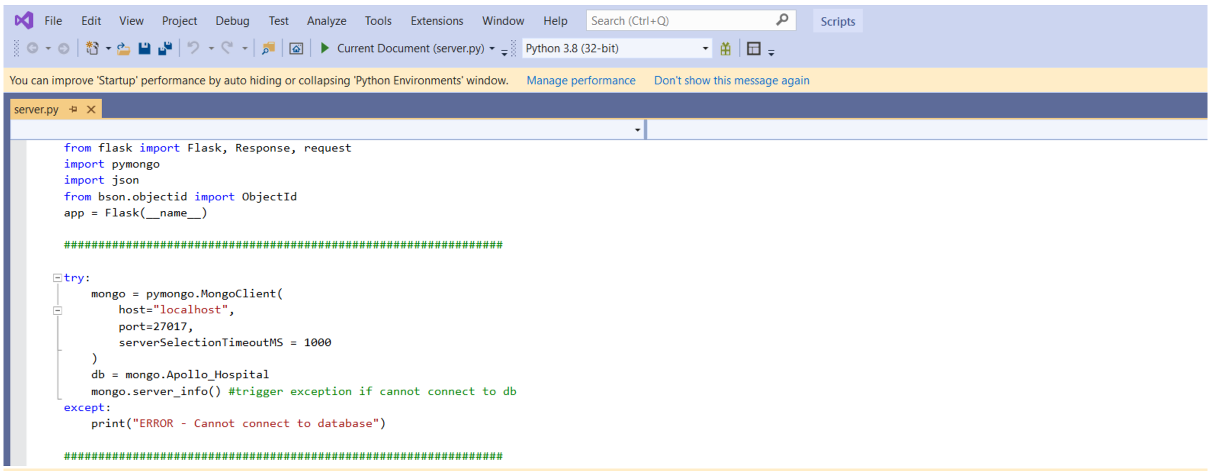
Name	Date modified	Type	Size
Include	12-07-2020 20:30	File folder	
Lib	12-07-2020 20:30	File folder	
Scripts	12-07-2020 21:54	File folder	
pyvenv.cfg	12-07-2020 20:30	CFG File	1 KB

5. Go to Scripts folder in CMD and run *activate* to activate the virtual environment
6. We will be inside mongo virtual environment.

7. Install flask using command *pip install flask*
8. Create server.py file which will contain our source code in the mongo folder
9. Install pymongo using command *pip install pymongo*
10. Run pip list to confirm if flask and pymongo is installed.

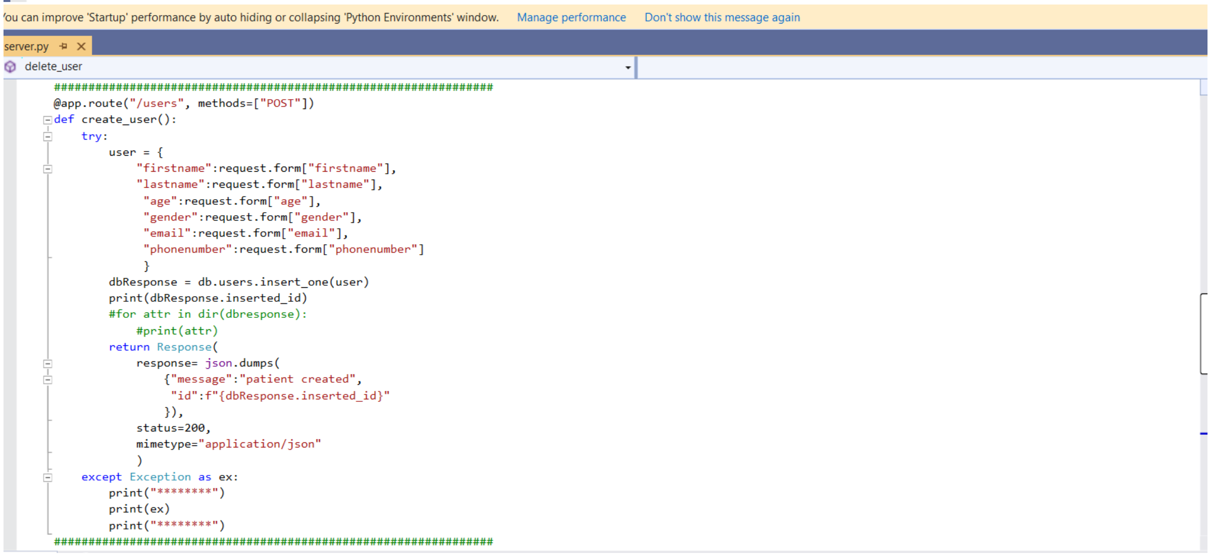
```
(mongo) C:\Program Files\MongoDB\flask\mongo>pip list
Package      Version
------------ -------
click        7.1.2
Flask        1.1.2
itsdangerous 1.1.0
Jinja2       2.11.2
MarkupSafe   1.1.1
pip          19.2.3
pymongo      3.10.1
setuptools   41.2.0
Werkzeug     1.0.1
WARNING: You are using pip version 19.2.3, however version 20.1.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.
```

11. Go to server.py file and create connection and database. Below is the snip of the source code. Here we will create database and collections.



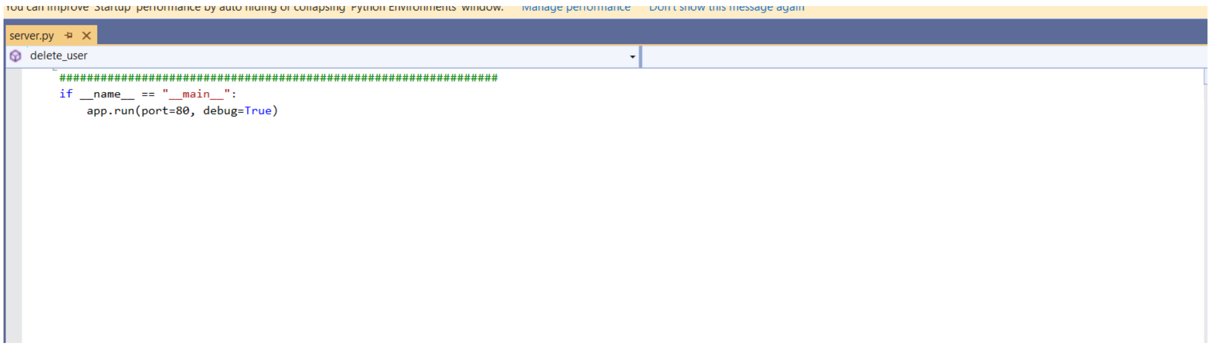
You can improve 'Startup' performance by auto hiding or collapsing 'Python Environments' window. Manage performance Don't show this message again

```python
from flask import Flask, Response, request
import pymongo
import json
from bson.objectid import ObjectId
app = Flask(__name__)

##################################################################

try:
    mongo = pymongo.MongoClient(
        host="localhost",
        port=27017,
        serverSelectionTimeoutMS = 1000
    )
    db = mongo.Apollo_Hospital
    mongo.server_info() #trigger exception if cannot connect to db
except:
    print("ERROR - Cannot connect to database")

##################################################################
```



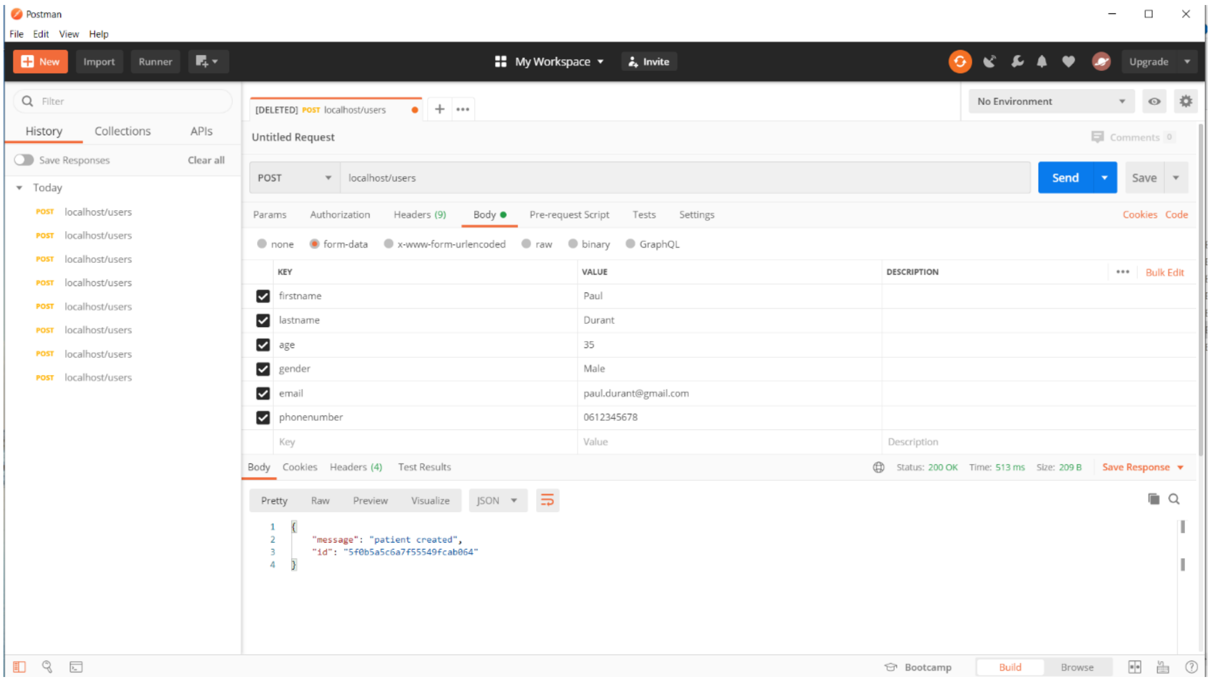
You can improve 'Startup' performance by auto hiding or collapsing 'Python Environments' window. Manage performance Don't show this message again

```python
##################################################################
@app.route("/users", methods=["POST"])
def create_user():
    try:
        user = {
            "firstname":request.form["firstname"],
            "lastname":request.form["lastname"],
             "age":request.form["age"],
             "gender":request.form["gender"],
             "email":request.form["email"],
             "phonenumber":request.form["phonenumber"]
             }
        dbResponse = db.users.insert_one(user)
        print(dbResponse.inserted_id)
        #for attr in dir(dbresponse):
            #print(attr)
        return Response(
            response= json.dumps(
                {"message":"patient created",
                 "id":f"{dbResponse.inserted_id}"
                }),
            status=200,
            mimetype="application/json"
            )
    except Exception as ex:
        print("********")
        print(ex)
        print("********")
##################################################################
```



```
    ###################################################################
    if __name__ == "__main__":
        app.run(port=80, debug=True)
```

12. Go to CMD and run server.py file and in another CMD run mongod.exe to run mongo daemon.
13. Go to Postman and run a POST query as in screenshot to create collections and enter data in database.



POST localhost/users

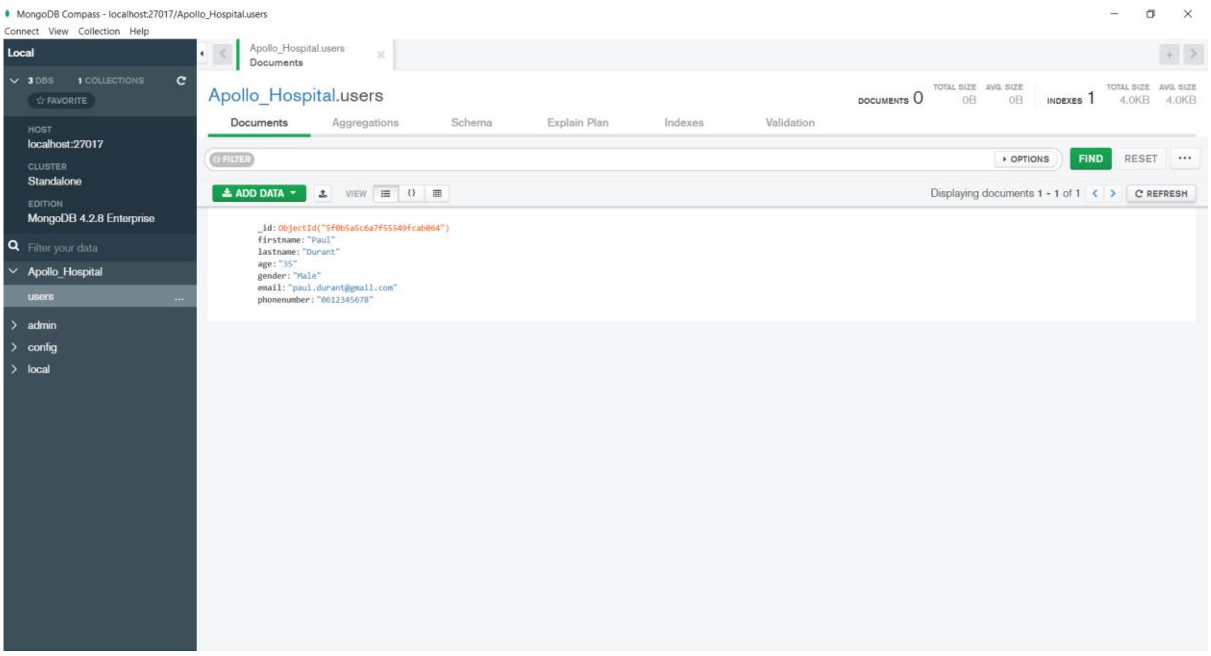
Body: form-data

KEY	VALUE	DESCRIPTION
firstname	Paul	
lastname	Durant	
age	35	
gender	Male	
email	paul.durant@gmail.com	
phonenumber	0612345678	

Status: 200 OK Time: 513 ms Size: 209 B

```
{
    "message": "patient created",
    "id": "5f0b5a5c6a7f55549fcab064"
}
```

14. We can go to compass and refresh databases to check if new patient entry is created



```
_id: ObjectId("5f0b5a5c6a7f55549fcab064")
firstname: "Paul"
lastname: "Durant"
age: "35"
gender: "Male"
email: "paul.durant@gmail.com"
phonenumber: "0612345678"
```

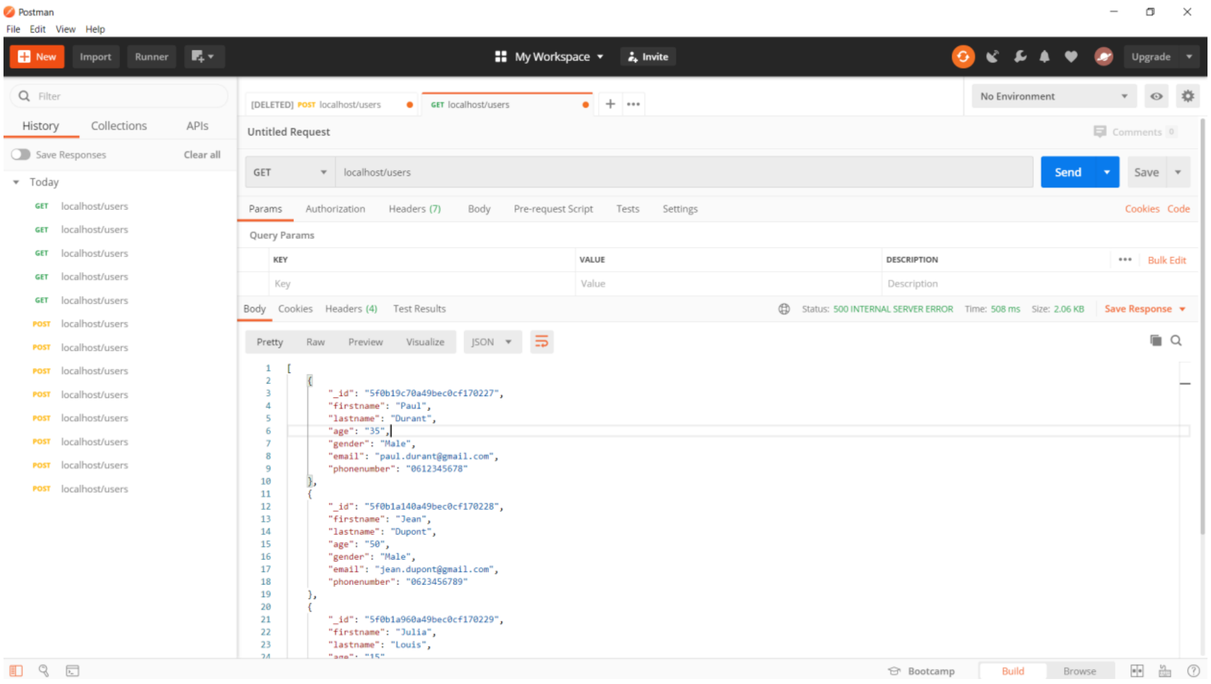
15. Similarly create the entire database.

Reading the data from database

1. Add below code in server.py file to read the data in postman

```
################################################################
@app.route("/users", methods=["GET"])
def get_some_users():
    try:
        data = list(db.users.find())
        for user in data:
            user["_id"] = str(user["_id"])
        return Response(
            response= json.dumps(data),
            status=500,
            mimetype="application/json")
    except Exception as ex:
        print(ex)
        return Response(response= json.dumps({"message":"Cannot read patient data"}), status=500, mimetype="application/json")

################################################################
```

2. Run server.py file in CMD and go to postman and create new connection and run GET query in localhost/users

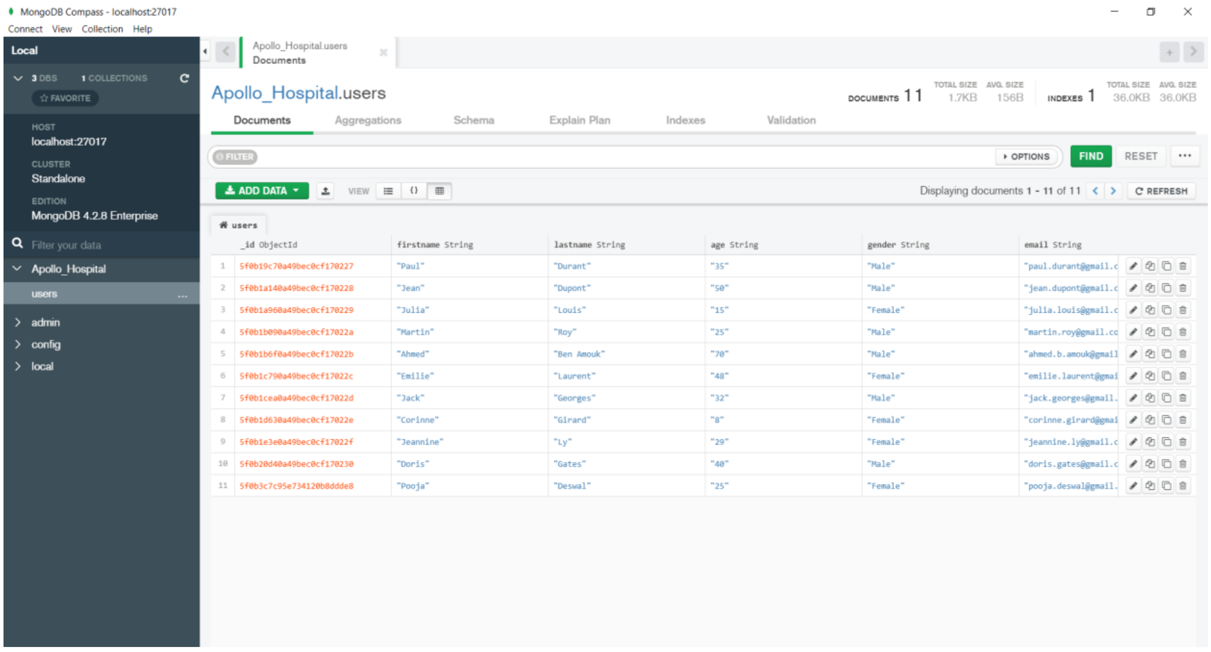


GET localhost/users

Status: 500 INTERNAL SERVER ERROR Time: 508 ms Size: 2.06 KB

```
[
    {
        "_id": "5f0b19c70a49bec0cf170227",
        "firstname": "Paul",
        "lastname": "Durant",
        "age": "35",
        "gender": "Male",
        "email": "paul.durant@gmail.com",
        "phonenumber": "0612345678"
    },
    {
        "_id": "5f0b1a140a49bec0cf170228",
        "firstname": "Jean",
        "lastname": "Dupont",
        "age": "50",
        "gender": "Male",
        "email": "jean.dupont@gmail.com",
        "phonenumber": "0623456789"
    },
    {
        "_id": "5f0b1a960a49bec0cf170229",
        "firstname": "Julia",
        "lastname": "Louis",
        "age": "15"
```

3. We can verify if we are able to fetch all data from compass



HOST localhost:27017

CLUSTER Standalone

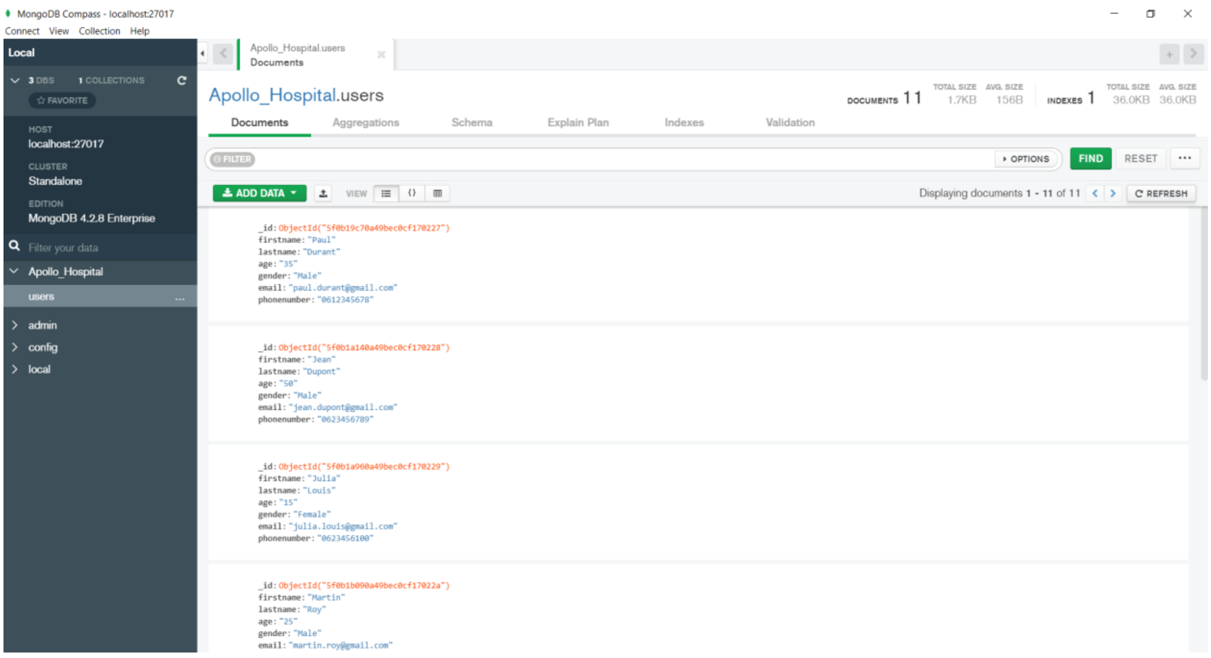
EDITION MongoDB 4.2.8 Enterprise

Apollo_Hospital.users

DOCUMENTS 11 TOTAL SIZE 1.7KB AVG. SIZE 156B INDEXES 1 TOTAL SIZE 36.0KB AVG. SIZE 36.0KB

Displaying documents 1 - 11 of 11

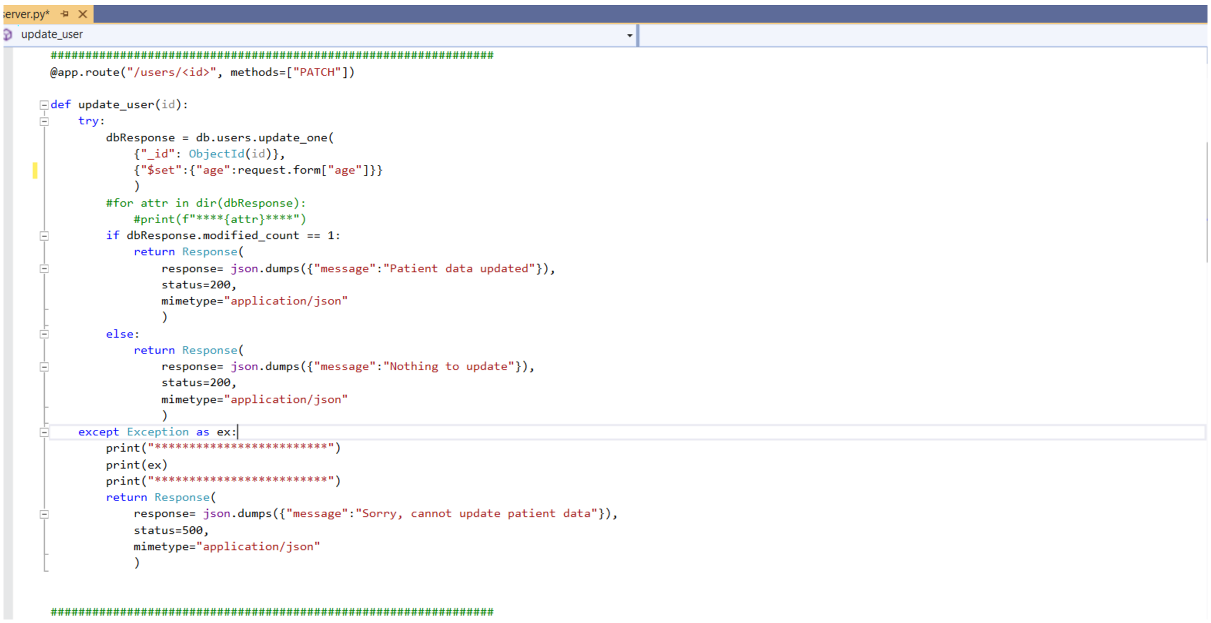
	_id ObjectId	firstname String	lastname String	age String	gender String	email String
1	5f0b19c70a49bec0cf170227	"Paul"	"Durant"	"35"	"Male"	"paul.durant@gmail.c
2	5f0b1a140a49bec0cf170228	"Jean"	"Dupont"	"50"	"Male"	"jean.dupont@gmail.c
3	5f0b1a960a49bec0cf170229	"Julia"	"Louis"	"15"	"Female"	"julia.louis@gmail.c
4	5f0b1b090a49bec0cf17022a	"Martin"	"Roy"	"25"	"Male"	"martin.roy@gmail.co
5	5f0b1b6f0a49bec0cf17022b	"Ahmed"	"Ben Amouk"	"70"	"Male"	"ahmed.b.amouk@gmail
6	5f0b1c790a49bec0cf17022c	"Emilie"	"Laurent"	"48"	"Female"	"emilie.laurent@gmai
7	5f0b1cea0a49bec0cf17022d	"Jack"	"Georges"	"32"	"Male"	"jack.georges@gmail.
8	5f0b1d630a49bec0cf17022e	"Corinne"	"Girard"	"8"	"Female"	"corinne.girard@gmai
9	5f0b1e3e0a49bec0cf17022f	"Jeannine"	"Ly"	"29"	"Female"	"jeannine.ly@gmail.c
10	5f0b20d40a49bec0cf170230	"Doris"	"Gates"	"40"	"Male"	"doris.gates@gmail.c
11	5f0b3c7c95e734120b8ddde8	"Pooja"	"Deswal"	"25"	"Female"	"pooja.deswal@gmail.



```
_id: ObjectId("5f0b19c70a49bec0cf170227")
firstname: "Paul"
lastname: "Durant"
age: "35"
gender: "Male"
email: "paul.durant@gmail.com"
phonenumber: "0612345678"

_id: ObjectId("5f0b1a140a49bec0cf170228")
firstname: "Jean"
lastname: "Dupont"
age: "50"
gender: "Male"
email: "jean.dupont@gmail.com"
phonenumber: "0623456789"

_id: ObjectId("5f0b1a960a49bec0cf170229")
firstname: "Julia"
lastname: "Louis"
age: "15"
gender: "Female"
email: "julia.louis@gmail.com"
phonenumber: "0623456100"

_id: ObjectId("5f0b1b090a49bec0cf17022a")
firstname: "Martin"
lastname: "Roy"
age: "25"
gender: "Male"
email: "martin.roy@gmail.com"
```

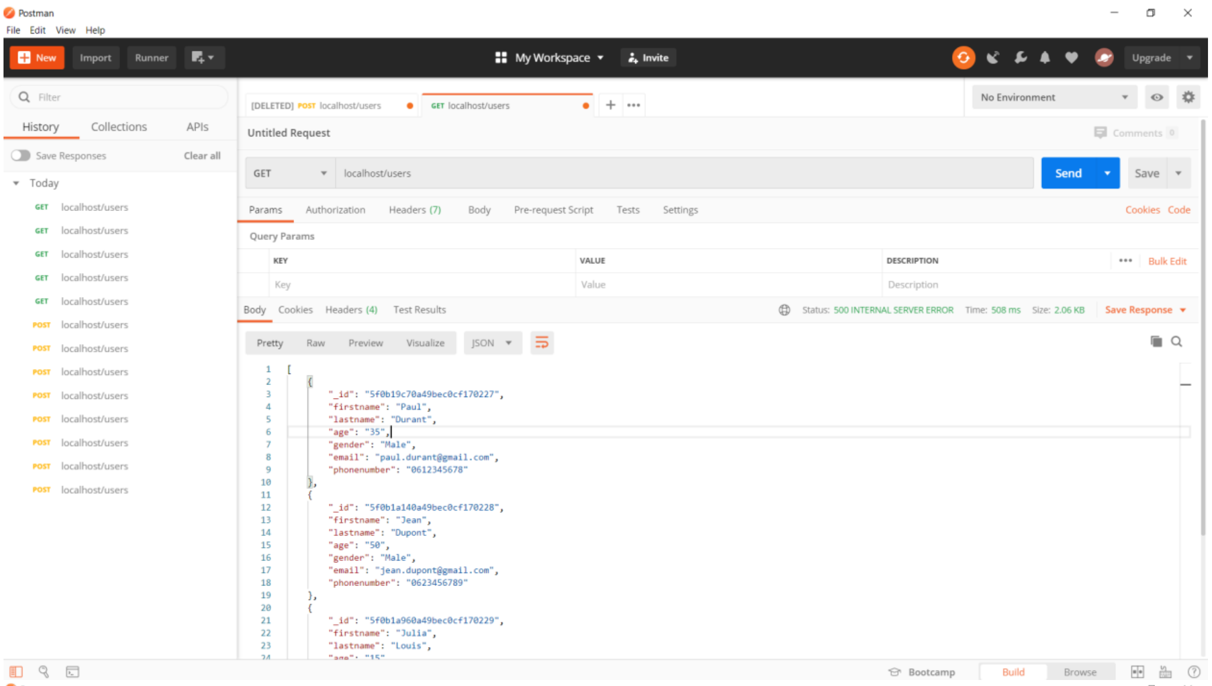
Updating Data in Database using ID:

1. Add below code in server.py to update age of the patients using their ID.



```
#################################################################
@app.route("/users/<id>", methods=["PATCH"])

def update_user(id):
    try:
        dbResponse = db.users.update_one(
            {"_id": ObjectId(id)},
            {"$set":{"age":request.form["age"]}}
            )
        #for attr in dir(dbResponse):
            #print(f"****{attr}****")
        if dbResponse.modified_count == 1:
            return Response(
                response= json.dumps({"message":"Patient data updated"}),
                status=200,
                mimetype="application/json"
                )
        else:
            return Response(
                response= json.dumps({"message":"Nothing to update"}),
                status=200,
                mimetype="application/json"
                )
    except Exception as ex:
        print("************************")
        print(ex)
        print("************************")
        return Response(
            response= json.dumps({"message":"Sorry, cannot update patient data"}),
            status=500,
            mimetype="application/json"
            )

#################################################################
```

2. We can verify if age is updated using PATCH method in postman



GET localhost/users
GET localhost/users
GET localhost/users
GET localhost/users
GET localhost/users
POST localhost/users
POST localhost/users
POST localhost/users
POST localhost/users
POST localhost/users
POST localhost/users
POST localhost/users
POST localhost/users

[DELETED] POST localhost/users | GET localhost/users

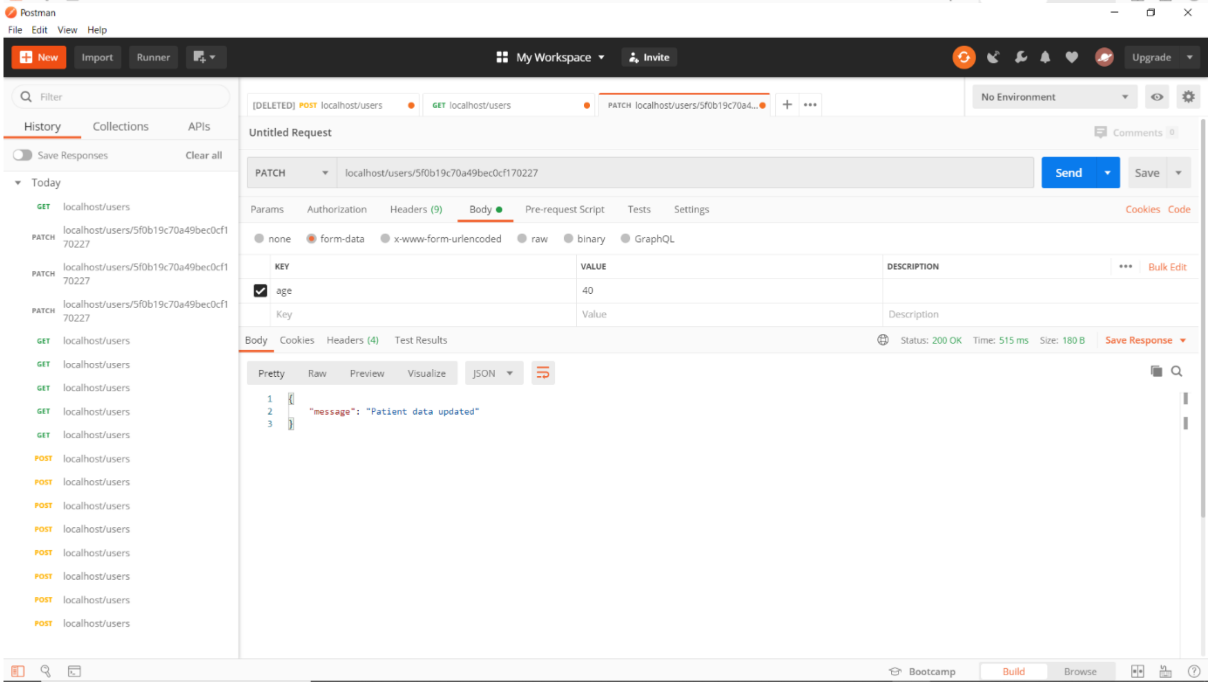
Untitled Request

GET localhost/users

KEY	VALUE	DESCRIPTION
Key	Value	Description

Status: 500 INTERNAL SERVER ERROR Time: 508 ms Size: 2.06 KB

```
[
    {
        "_id": "5f0b19c70a49bec0cf170227",
        "firstname": "Paul",
        "lastname": "Durant",
        "age": "35",
        "gender": "Male",
        "email": "paul.durant@gmail.com",
        "phonenumber": "0612345678"
    },
    {
        "_id": "5f0b1a140a49bec0cf170228",
        "firstname": "Jean",
        "lastname": "Dupont",
        "age": "50",
        "gender": "Male",
        "email": "jean.dupont@gmail.com",
        "phonenumber": "0623456789"
    },
    {
        "_id": "5f0b1a960a49bec0cf170229",
        "firstname": "Julia",
        "lastname": "Louis",
        "age": "15"
```



GET localhost/users
PATCH localhost/users/5f0b19c70a49bec0cf170227
PATCH localhost/users/5f0b19c70a49bec0cf170227
PATCH localhost/users/5f0b19c70a49bec0cf170227
GET localhost/users
GET localhost/users
GET localhost/users
GET localhost/users
GET localhost/users
POST localhost/users
POST localhost/users
POST localhost/users
POST localhost/users
POST localhost/users
POST localhost/users
POST localhost/users
POST localhost/users

[DELETED] POST localhost/users | GET localhost/users | PATCH localhost/users/5f0b19c70a4...

Untitled Request

PATCH localhost/users/5f0b19c70a49bec0cf170227

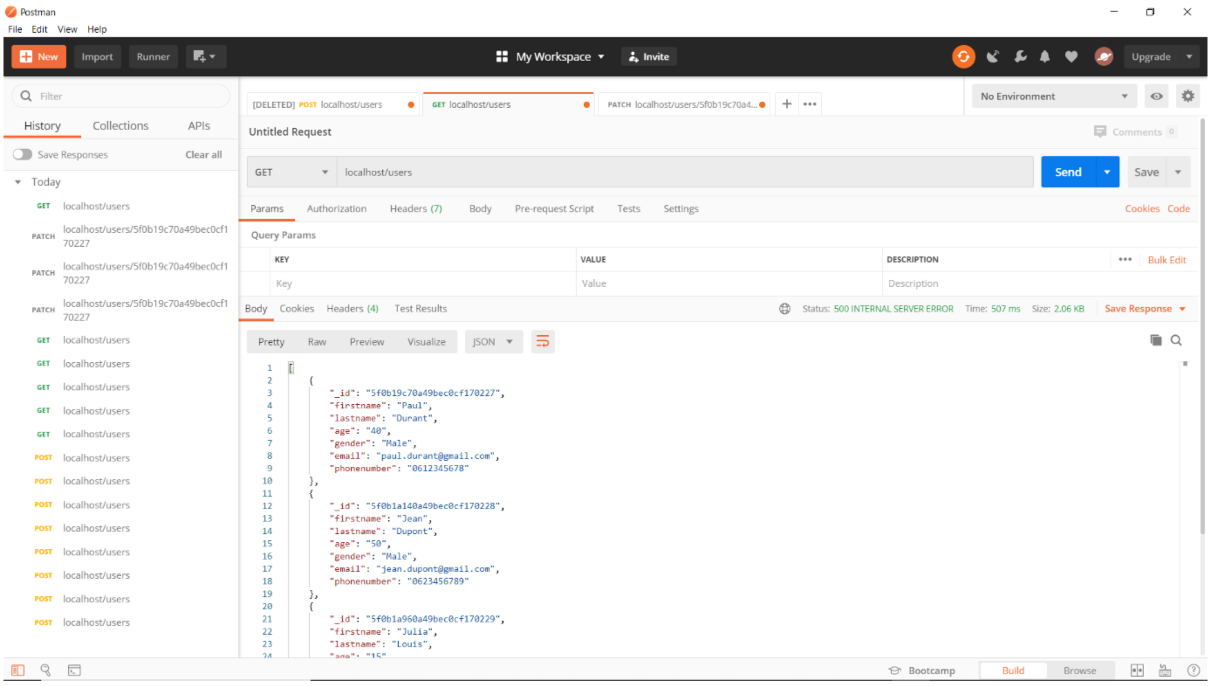
Body: form-data

	KEY	VALUE	DESCRIPTION
✓	age	40	
	Key	Value	Description

Status: 200 OK Time: 515 ms Size: 180 B

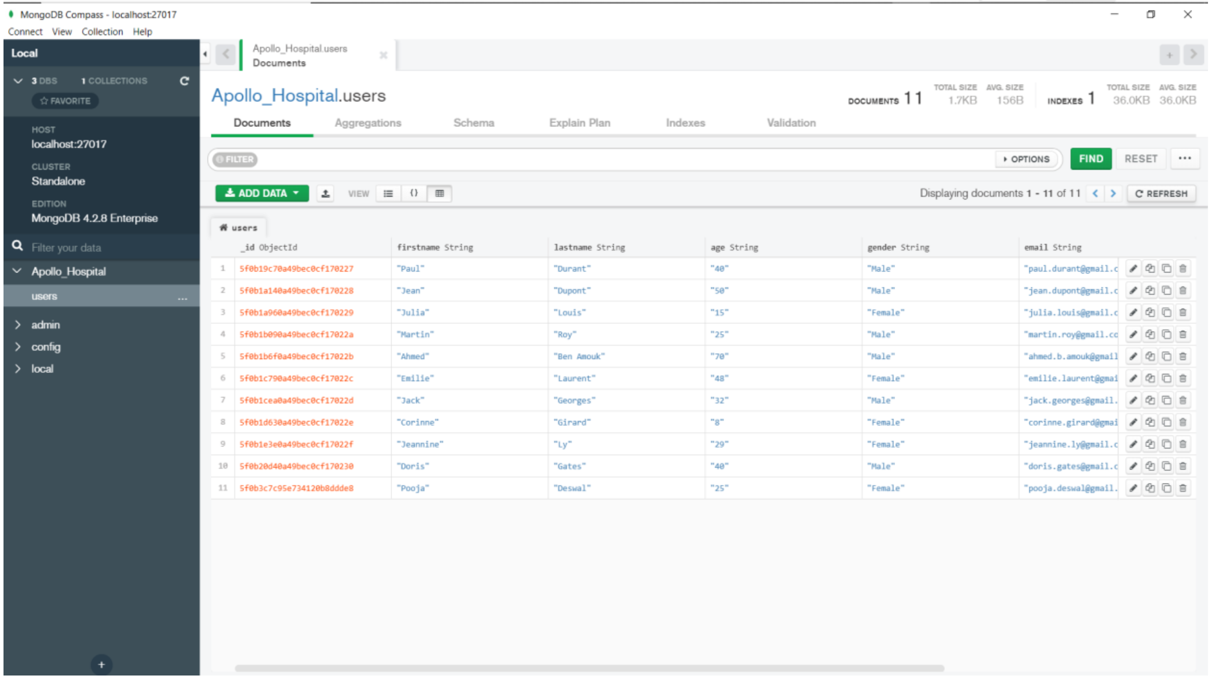
```
{
    "message": "Patient data updated"
}
```

3. After update if we again run GET method, we will see updated age of patient


GET localhost/users

Status: 500 INTERNAL SERVER ERROR Time: 507 ms Size: 2.06 KB

```
[
    {
        "_id": "5f0b19c70a49bec0cf170227",
        "firstname": "Paul",
        "lastname": "Durant",
        "age": "40",
        "gender": "Male",
        "email": "paul.durant@gmail.com",
        "phonenumber": "0612345678"
    },
    {
        "_id": "5f0b1a140a49bec0cf170228",
        "firstname": "Jean",
        "lastname": "Dupont",
        "age": "50",
        "gender": "Male",
        "email": "jean.dupont@gmail.com",
        "phonenumber": "0623456789"
    },
    {
        "_id": "5f0b1a960a49bec0cf170229",
        "firstname": "Julia",
        "lastname": "Louis",
        "age": "15"
```

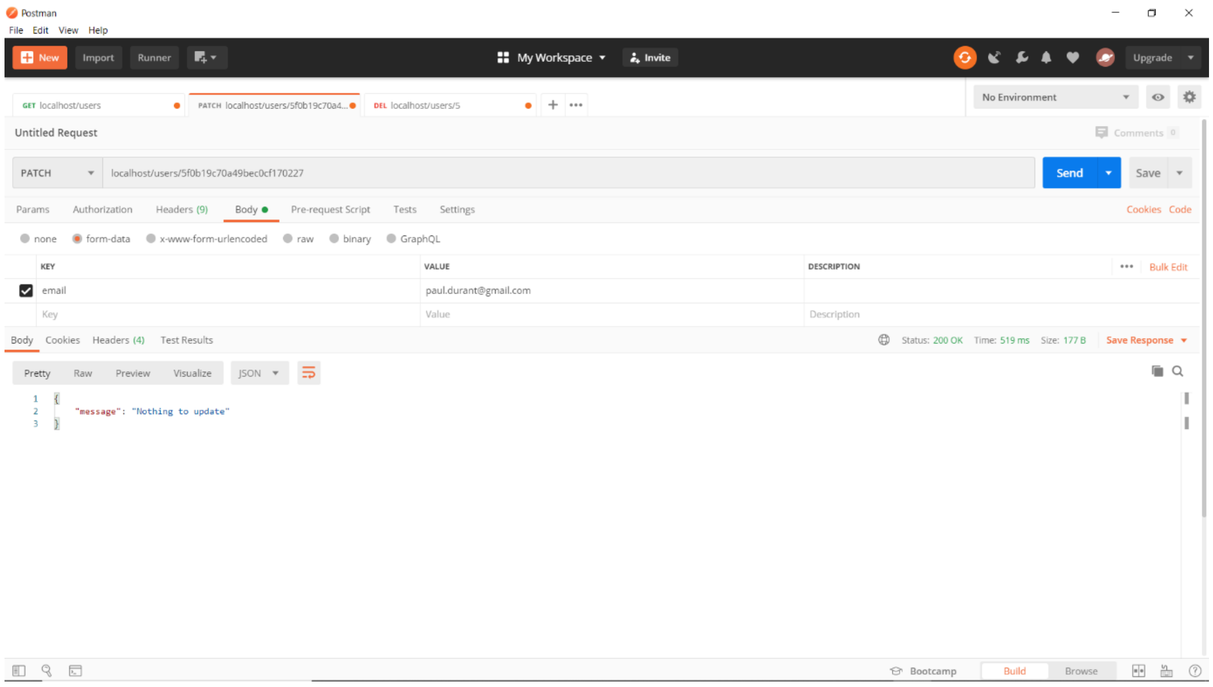

Apollo_Hospital.users

DOCUMENTS 11 TOTAL SIZE 1.7KB AVG. SIZE 156B INDEXES 1 TOTAL SIZE 36.0KB AVG. SIZE 36.0KB

Displaying documents 1 - 11 of 11

	_id ObjectId	firstname String	lastname String	age String	gender String	email String
1	5f0b19c70a49bec0cf170227	"Paul"	"Durant"	"40"	"Male"	"paul.durant@gmail.c
2	5f0b1a140a49bec0cf170228	"Jean"	"Dupont"	"50"	"Male"	"jean.dupont@gmail.c
3	5f0b1a960a49bec0cf170229	"Julia"	"Louis"	"15"	"Female"	"julia.louis@gmail.c
4	5f0b1b090a49bec0cf17022a	"Martin"	"Roy"	"25"	"Male"	"martin.roy@gmail.cc
5	5f0b1b6f0a49bec0cf17022b	"Ahmed"	"Ben Amouk"	"70"	"Male"	"ahmed.b.amouk@gmail
6	5f0b1c790a49bec0cf17022c	"Emilie"	"Laurent"	"48"	"Female"	"emilie.laurent@gmai
7	5f0b1cea0a49bec0cf17022d	"Jack"	"Georges"	"32"	"Male"	"jack.georges@gmail.
8	5f0b1d630a49bec0cf17022e	"Corinne"	"Girard"	"8"	"Female"	"corinne.girard@gmai
9	5f0b1e3e0a49bec0cf17022f	"Jeannine"	"Ly"	"29"	"Female"	"jeannine.ly@gmail.c
10	5f0b20d40a49bec0cf170230	"Doris"	"Gates"	"40"	"Male"	"doris.gates@gmail.c
11	5f0b3c7c95e734120b8ddde8	"Pooja"	"Deswal"	"25"	"Female"	"pooja.deswal@gmail.

4. In case we try to update the same data, example the same email present already in database we get below message



Untitled Request

PATCH localhost/users/5f0b19c70a49bec0cf170227

none [✓ form-data] x-www-form-urlencoded raw binary GraphQL

KEY	VALUE	DESCRIPTION
✓ email	paul.durant@gmail.com	
Key	Value	Description

Status: 200 OK Time: 519 ms Size: 177 B

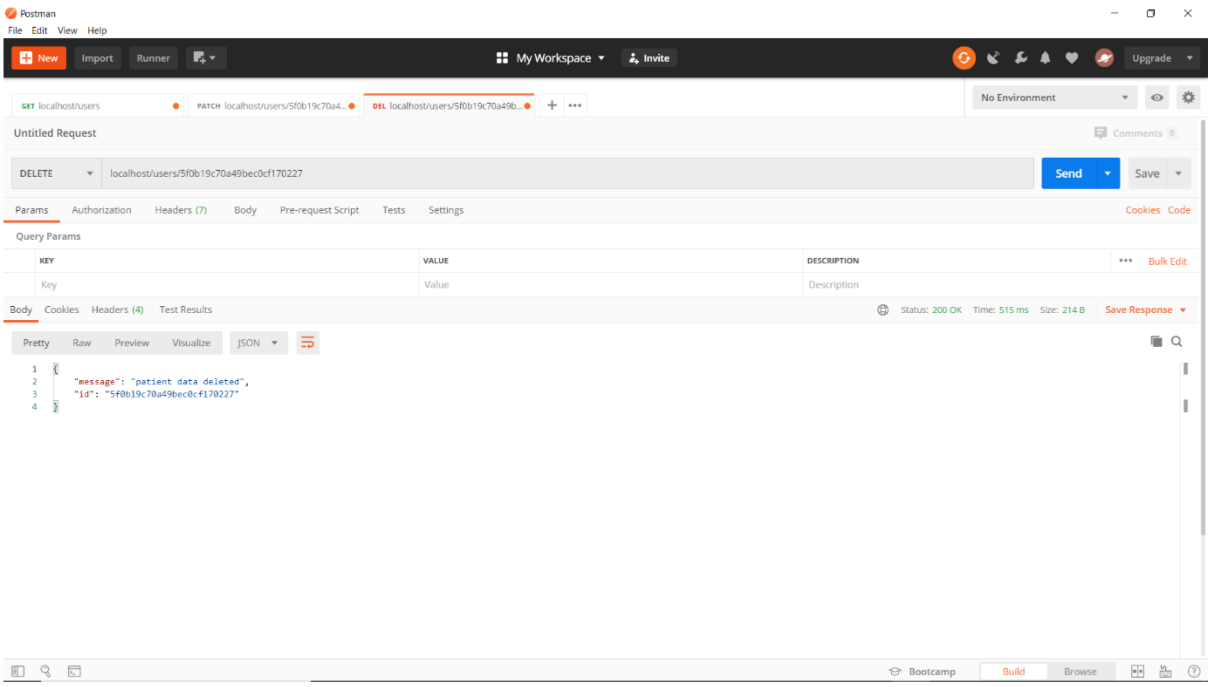
```
{
    "message": "Nothing to update"
}
```

Deleting an entry from database

1. Add below code in server.py file and run it in CMD

```
###################################################################
@app.route("/users/<id>", methods=["DELETE"])

def delete_user(id):
    try:
        dbResponse = db.users.delete_one({"_id":ObjectId(id)})
        #for attr in dir(dbresponse):
            #print(f"****{attr}****")
        if dbResponse.deleted_count == 1:
            return Response(
                response= json.dumps({"message":"patient data deleted",
                 "id":f"{id}"}),
                status=200,
                mimetype="application/json"
                )
        return Response(
            response= json.dumps(
                {"message":"patient data not available in database",
                 "id":f"{id}"
                }),
            status=200,
            mimetype="application/json"
            )

    except Exception as ex:
        print("********")
        print(ex)
        print("********")
        return Response(response= json.dumps({"message":"Cannot delete patient data"}), status=500, mimetype="application/json")
###################################################################
if __name__ == "__main__":
    app.run(port=80, debug=True)
```

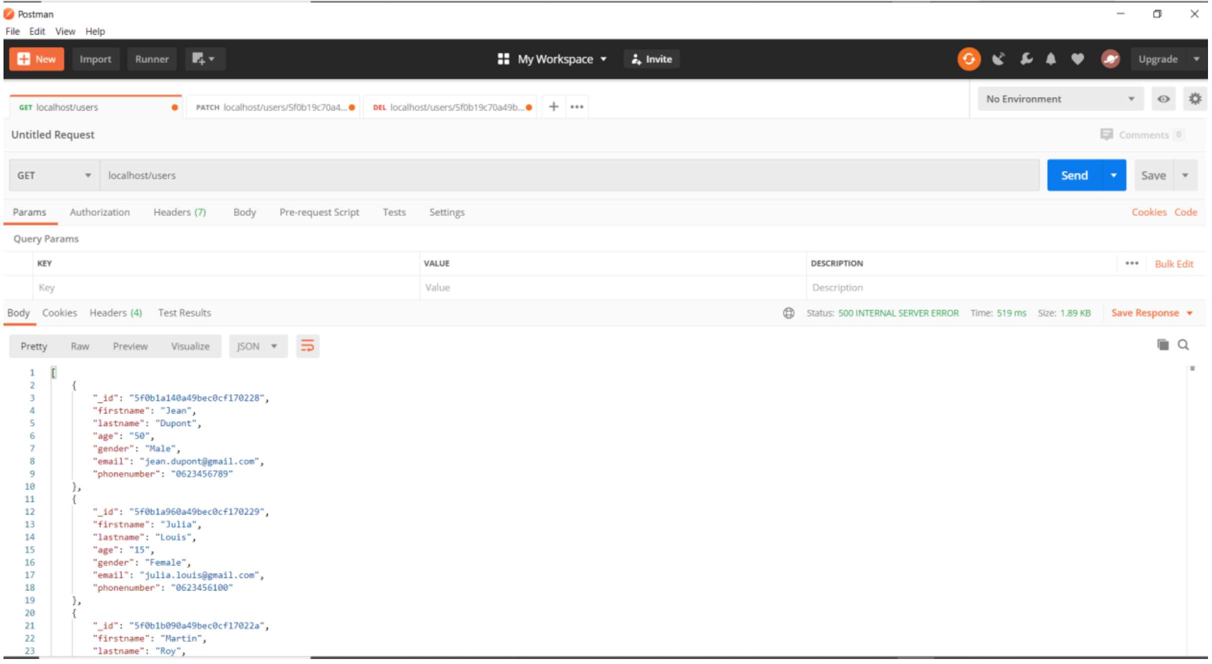
2. Go to postman and try to run delete query using patient ID



DELETE localhost/users/5f0b19c70a49bec0cf170227

Status: 200 OK Time: 515 ms Size: 214 B

```
{
    "message": "patient data deleted",
    "id": "5f0b19c70a49bec0cf170227"
}
```

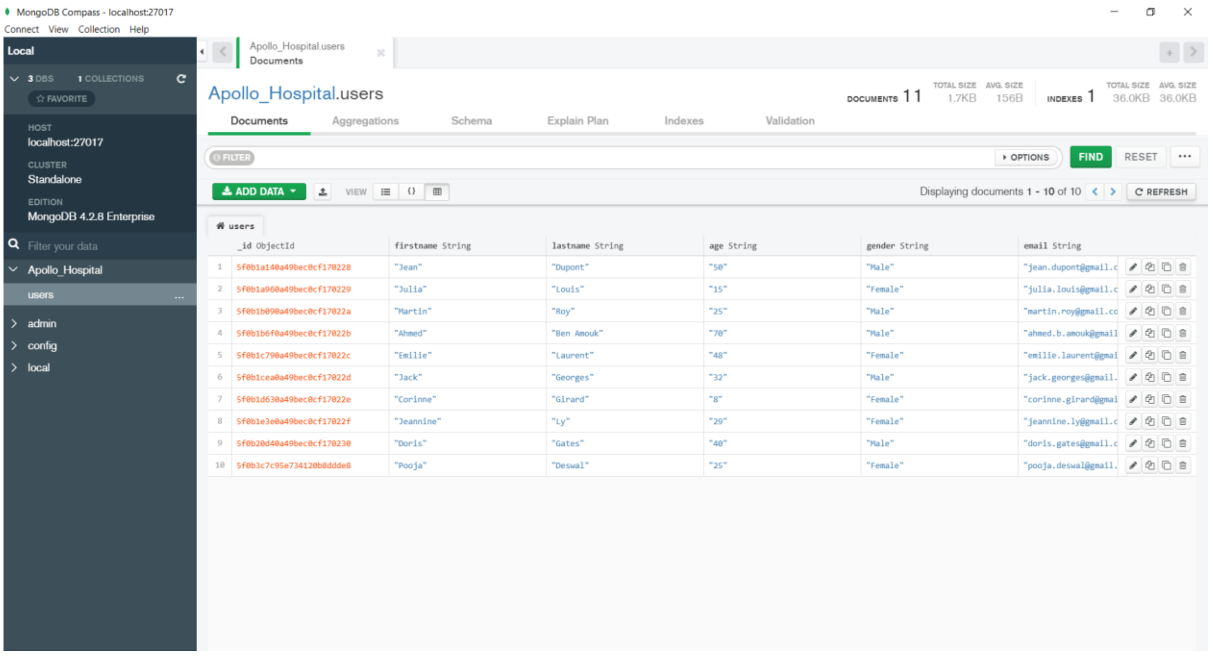


GET localhost/users

Status: 500 INTERNAL SERVER ERROR Time: 519 ms Size: 1.89 KB

```
[
    {
        "_id": "5f0b1a140a49bec0cf170228",
        "firstname": "Jean",
        "lastname": "Dupont",
        "age": "50",
        "gender": "Male",
        "email": "jean.dupont@gmail.com",
        "phonenumber": "0623456789"
    },
    {
        "_id": "5f0b1a960a49bec0cf170229",
        "firstname": "Julia",
        "lastname": "Louis",
        "age": "15",
        "gender": "Female",
        "email": "julia.louis@gmail.com",
        "phonenumber": "0623456100"
    },
    {
        "_id": "5f0b1b090a49bec0cf17022a",
        "firstname": "Martin",
        "lastname": "Roy",
```

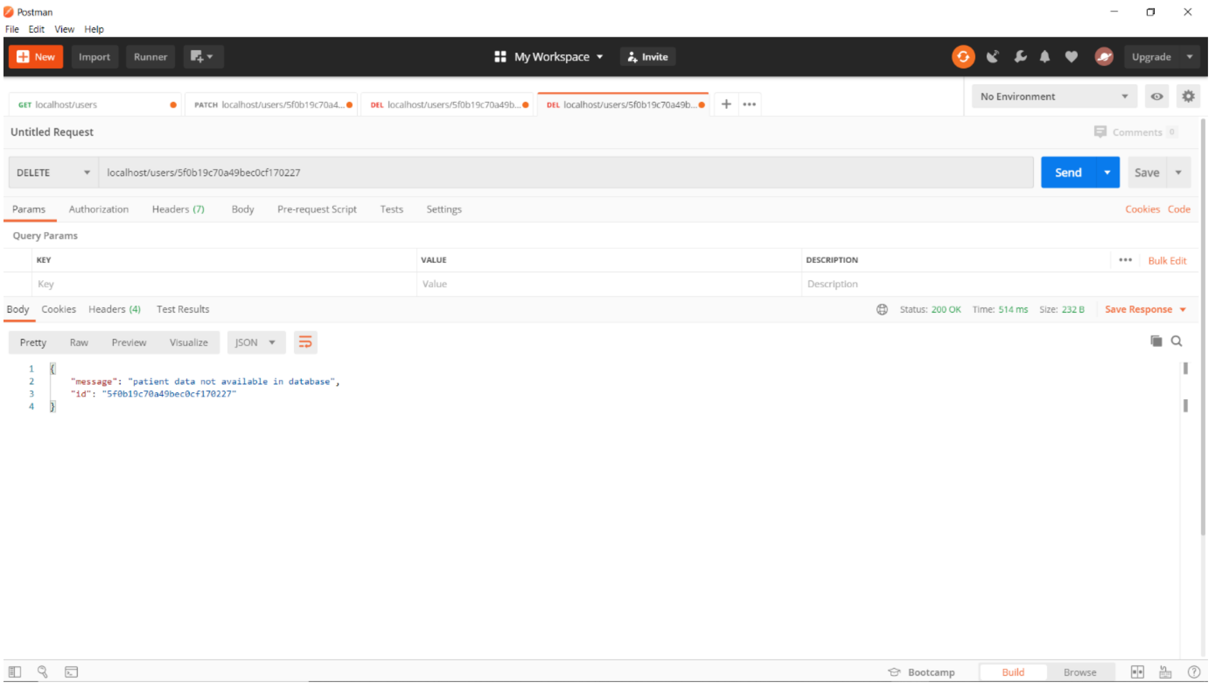
3. We go to compass to verify of patient entry is deleted



users

	_id ObjectId	firstname String	lastname String	age String	gender String	email String
1	5f0b1a140a49bec0cf170228	"Jean"	"Dupont"	"50"	"Male"	"jean.dupont@gmail.c
2	5f0b1a960a49bec0cf170229	"Julia"	"Louis"	"15"	"Female"	"julia.louis@gmail.c
3	5f0b1b090a49bec0cf17022a	"Martin"	"Roy"	"25"	"Male"	"martin.roy@gmail.co
4	5f0b1b6f0a49bec0cf17022b	"Ahmed"	"Ben Amouk"	"70"	"Male"	"ahmed.b.amouk@gmail
5	5f0b1c790a49bec0cf17022c	"Emilie"	"Laurent"	"48"	"Female"	"emilie.laurent@gmai
6	5f0b1cea0a49bec0cf17022d	"Jack"	"Georges"	"32"	"Male"	"jack.georges@gmail.
7	5f0b1d630a49bec0cf17022e	"Corinne"	"Girard"	"8"	"Female"	"corinne.girard@gmai
8	5f0b1e3e0a49bec0cf17022f	"Jeannine"	"Ly"	"29"	"Female"	"jeannine.ly@gmail.c
9	5f0b20d40a49bec0cf170230	"Doris"	"Gates"	"40"	"Male"	"doris.gates@gmail.c
10	5f0b3c7c95e734120b8ddde8	"Pooja"	"Deswal"	"25"	"Female"	"pooja.deswal@gmail.

4. In case we try to delete patient, whose ID is not present in database we get message as in below screenshot



DELETE localhost/users/5f0b19c70a49bec0cf170227

Status: 200 OK Time: 514 ms Size: 232 B

```
{
    "message": "patient data not available in database",
    "id": "5f0b19c70a49bec0cf170227"
}
```

Please find the source code, server.py file in below Github repository

https://github.com/PoojaDeswal94/NOSQLAssignment