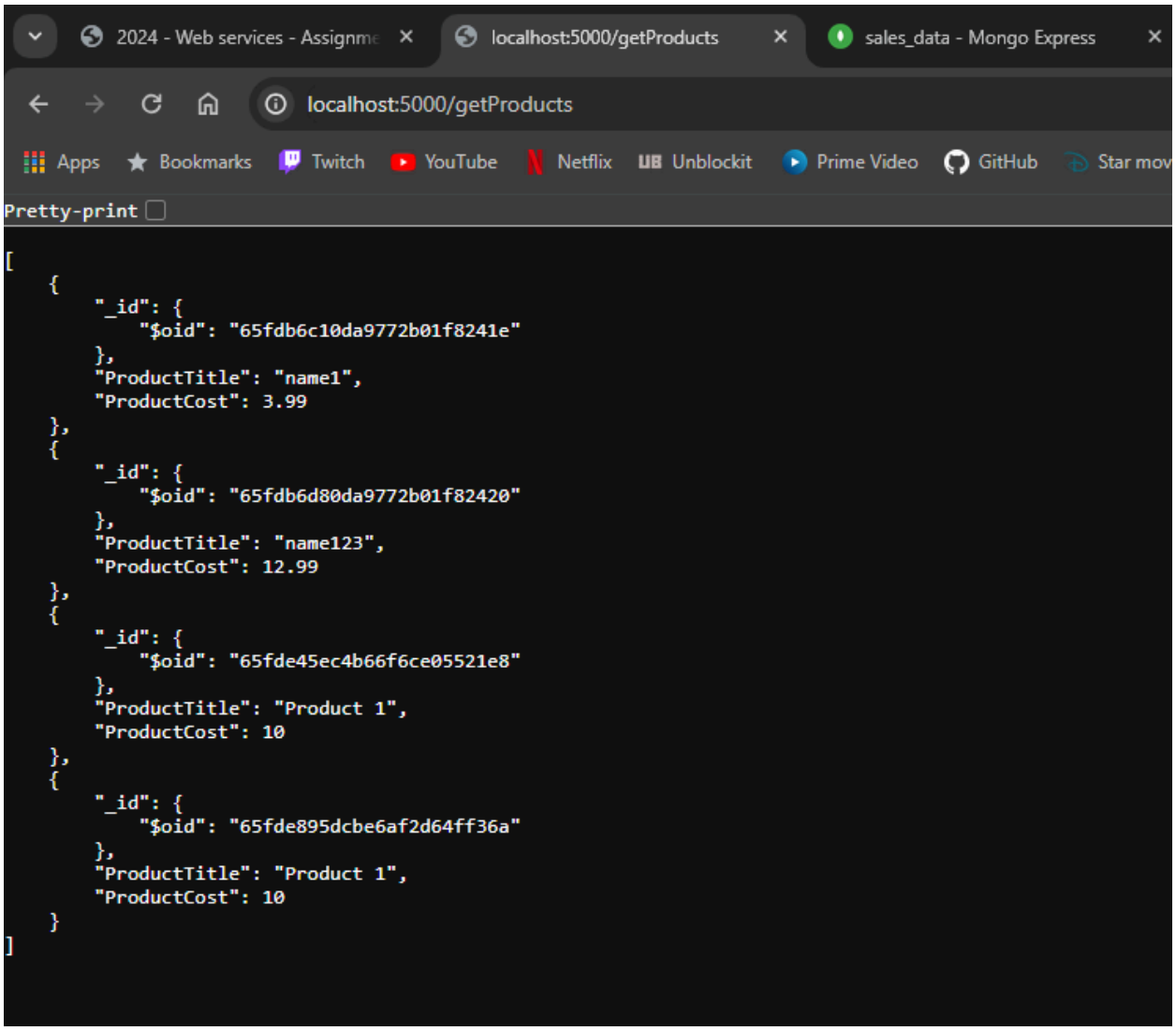
# Assignment 1

By B00134706 - Sarah McCabe

## Part 1- Restful API

### /getProducts

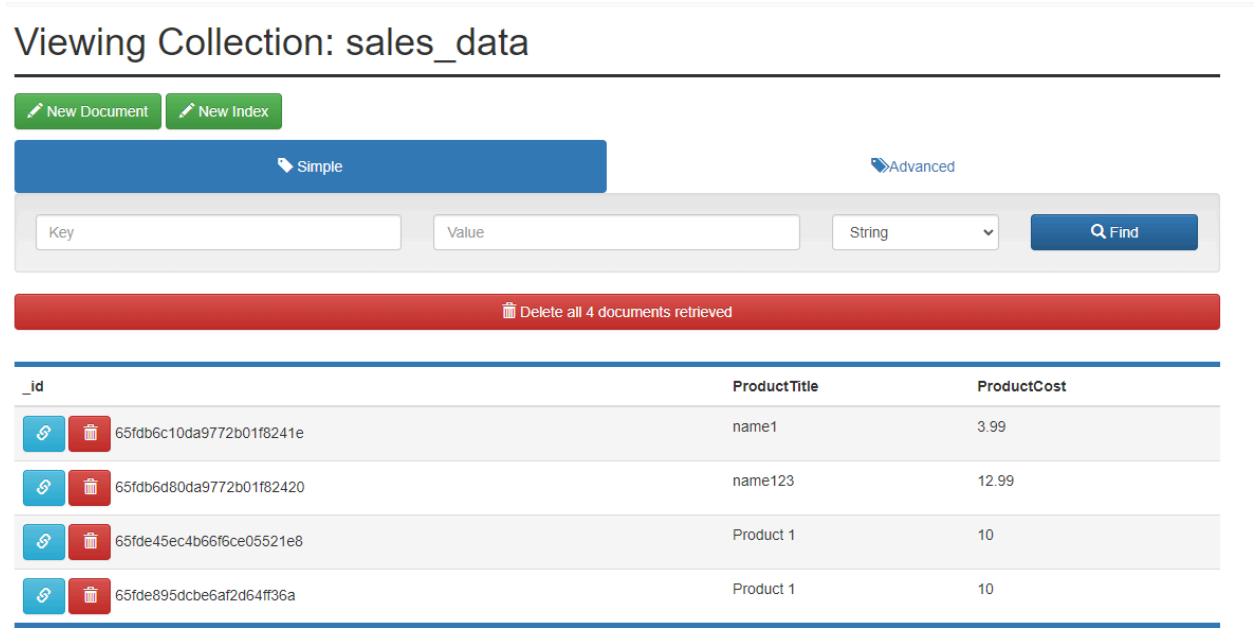
For this page all the products in the mongo database are taken and dumped out to the page as a JSON. Below is a screenshot of the results of this page.





```
[
    {
        "_id": {
            "$oid": "65fdb6c10da9772b01f8241e"
        },
        "ProductTitle": "name1",
        "ProductCost": 3.99
    },
    {
        "_id": {
            "$oid": "65fdb6d80da9772b01f82420"
        },
        "ProductTitle": "name123",
        "ProductCost": 12.99
    },
    {
        "_id": {
            "$oid": "65fde45ec4b66f6ce05521e8"
        },
        "ProductTitle": "Product 1",
        "ProductCost": 10
    },
    {
        "_id": {
            "$oid": "65fde895dcbe6af2d64ff36a"
        },
        "ProductTitle": "Product 1",
        "ProductCost": 10
    }
]
```

Here is a screenshot of the mongoDB it runs on localhost port 8081 and is set up in a docker container using a docker yaml file for the project. The items from the screenshot above are pulled from the database and can be seen in the screenshot below.



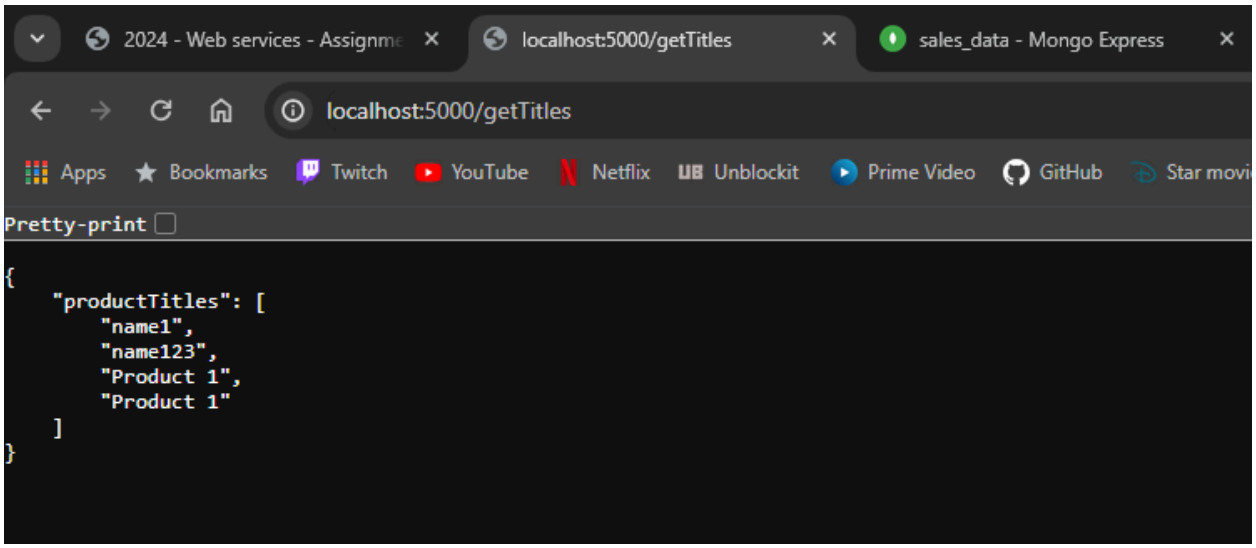

## Rename Collection

Here is the code in the simple api file that runs this page. The API is on a github repo that is pulled down into the Jenkins. Here is a link to the repo [https://github.com/Smccb/Jenkins/blob/main/sample_api.py](https://github.com/Smccb/Jenkins/blob/main/sample_api.py)

```
class GetProducts(Resource):
    def get(self):
        results = dumps(collection.find())
        return json.loads(results)
api.add_resource(GetProducts, '/getProducts')
```

## /getTitles

This page gets just the titles from the mongoDB and uses GraphQL query to preform this below is a screenshot of the code working along with the code that is in the sampleAPI file and preforms this action.



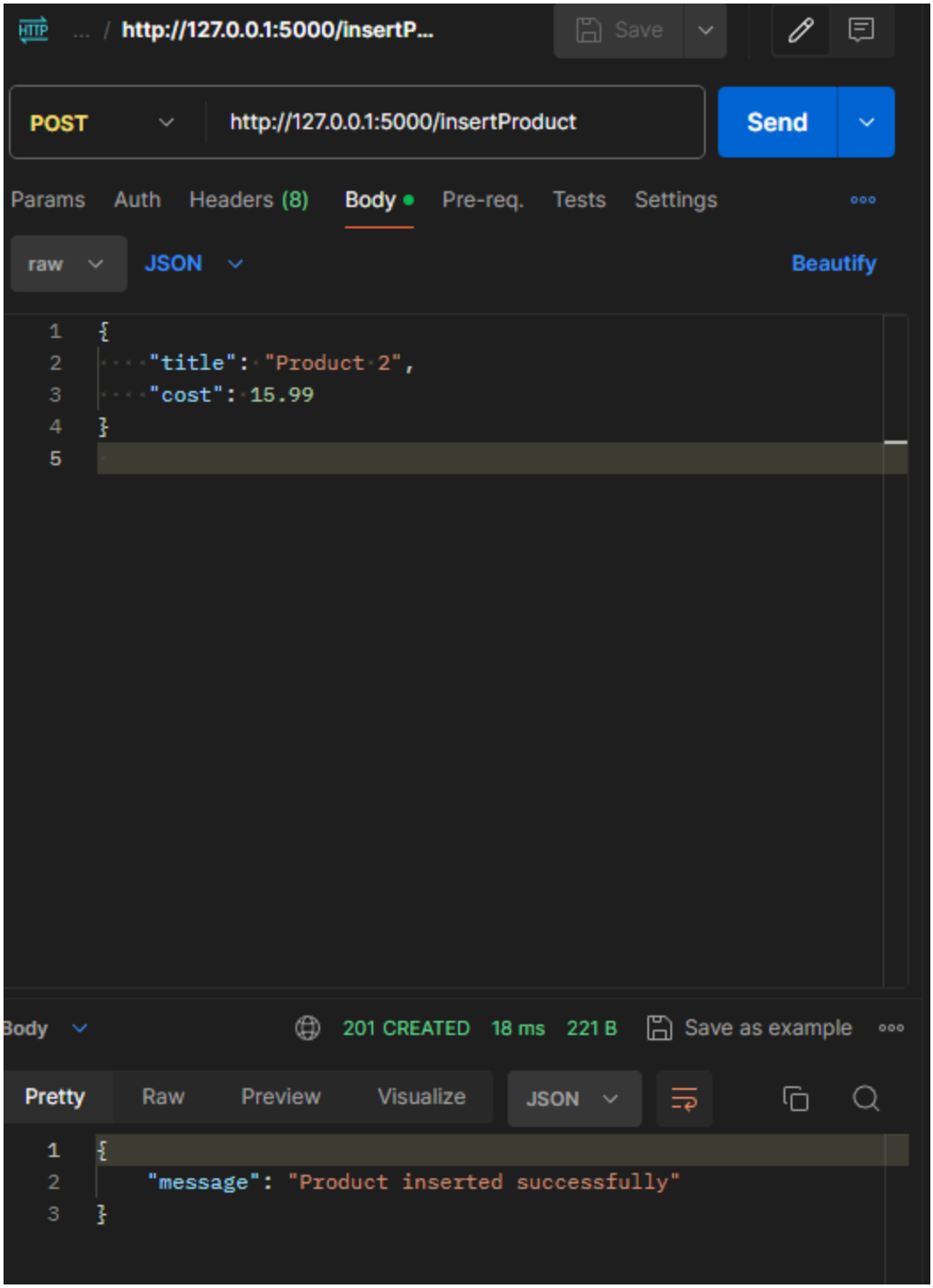

Screenshot of the mongoDB showing all the product titles it currently has.

| ProductTitle |
| --- |
| name1 |
| name123 |
| Product 1 |
| Product 1 |

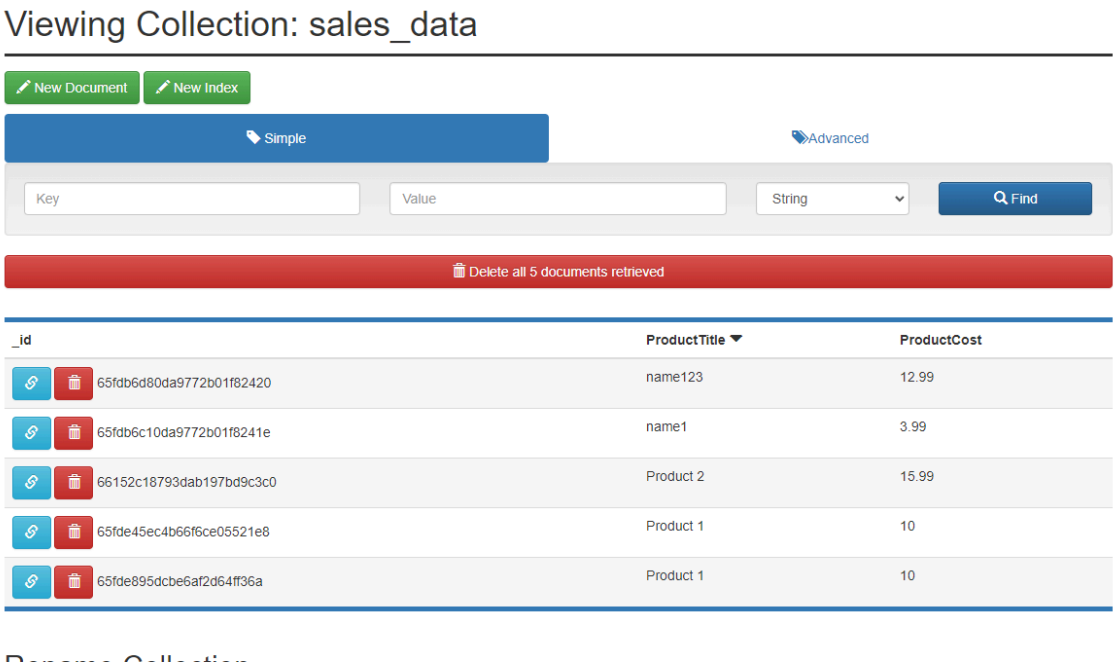
```python
class GetTitles(Resource):
    def get(self):
        # Running a query on the data
        schema = graphene.Schema(query=Query)
        query = """
        {
            productTitles
        }
        """
        result = schema.execute(query)
        return json.loads(json.dumps(result.data))
api.add_resource(GetTitles, '/getTitles')
```

## /insertProduct

For this page, it accepts posts and pushes the items posted to the mongoDB. This was preformed using postman and a screenshot of an example postman post can be seen in the screenshot below.




As you can seen in the screenshot above the product was inserted and because it was you can now see the new product in the mongoDB database in the screenshot below.




Here is the code that was used in the sample_api.py file to preform the tasks for this page.

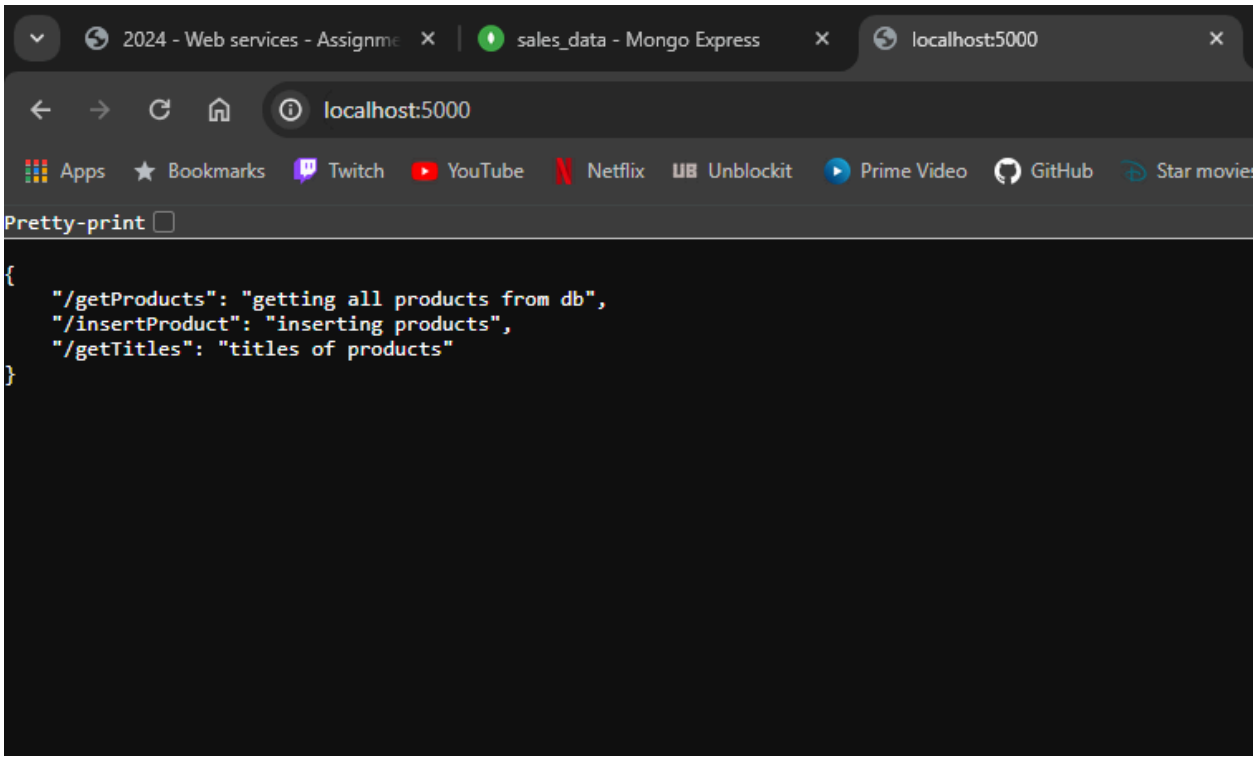
```
class InsertProducts(Resource):
    def post(self):

        # Retrieve data from the request
        data = request.get_json()
        title = data.get("title")
        cost = data.get("cost")

        # Insert the product into the database
        new_record = {"ProductTitle": title, "ProductCost": cost}
        collection.insert_one(new_record)

        # Return a response
        return {"message": "Product inserted successfully"}, 201
api.add_resource(InsertProducts, '/insertProduct')
```

/ - index/root page



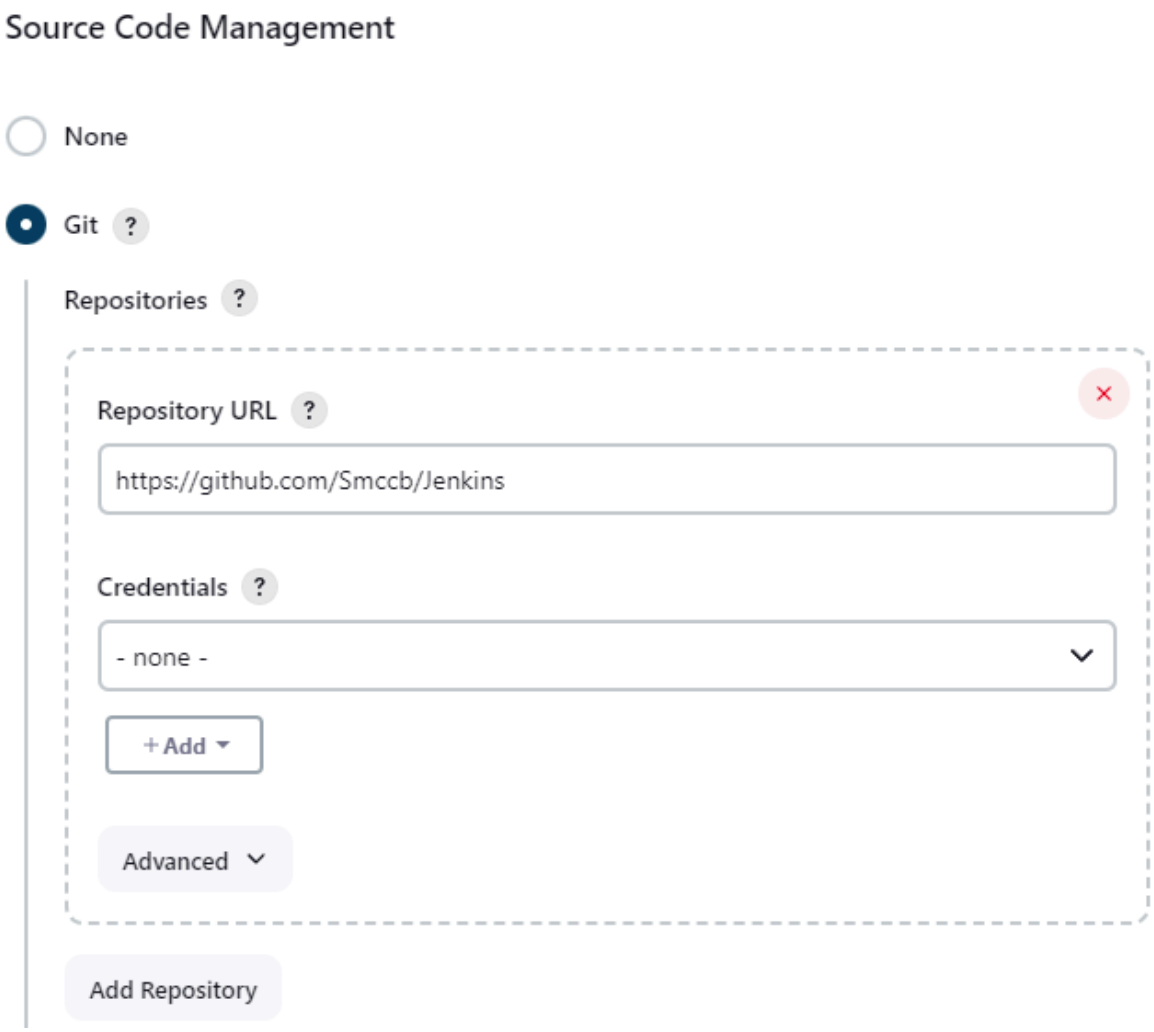

```python
class Root(Resource):
    def get(self):
        return {'/getProducts': 'getting all products from db',
                '/insertProduct': 'inserting products',
                '/getTitles': 'titles of products'}

api.add_resource(Root, '/')
```

# Part 2 - DevOps

1. Pulling code from GitHub

   Here is a sample screenshot of the code being pulled from the github as well as running the simpleAPI.py file in the background it stops by default at the end before I stopped the docker container same with the test.py tests files.




```
C:\ProgramData\Jenkins\.jenkins\workspace\test2\.git # timeout=10
Fetching changes from the remote Git repository
 > git.exe config remote.origin.url https://github.com/Smccb/Jenkins # timeout=10
Fetching upstream changes from https://github.com/Smccb/Jenkins
```

The directory for the server after showing the cloned items from the repo

```
 Directory of C:\ProgramData\Jenkins\.jenkins\workspace\test2

22/03/2024  21:22    <DIR>          .
21/03/2024  13:47    <DIR>          ..
21/03/2024  23:17               538 compose.yaml
21/03/2024  13:47                 9 README.md
22/03/2024  21:22             2,258 sample_api.py
21/03/2024  23:17             1,437 test.py
09/04/2024  13:06               371 test_saved.log
               5 File(s)          4,613 bytes
               2 Dir(s)  365,437,808,640 bytes free
```

```
C:\ProgramData\Jenkins\.jenkins\workspace\test2>start cmd /c "python sample_api.py"
```

## 2. Turning on servers

Here is an example of the mongoDB server being turned on and created from the docker compose yml file in the online repo that was cloned onto the jenkins server.

```
C:\ProgramData\Jenkins\.jenkins\workspace\test2>docker-compose up -d
 Container test2-mongo-express-1  Created
 Container test2-mongo-1  Created
 Container test2-mongo-express-1  Starting
 Container test2-mongo-1  Starting
 Container test2-mongo-express-1  Started
 Container test2-mongo-1  Started
```

## 3. Unit tests

Here is the code for the unit tests they check the /getProducts, /getTitles and the / or root urls to check that the correct items are displaying the correct items on each page

The getProducts page expected to have at least one product with the name1 name, this is the same for getTitles page.

Console Output

```
C:\ProgramData\Jenkins\.jenkins\workspace\test2>python test.py
[
    {
        "_id": {
            "$oid": "65fdb6c10da9772b01f8241e"
        },
        "ProductTitle": "name1",
        "ProductCost": 3.99
    },
    {
        "_id": {
            "$oid": "65fdb6d80da9772b01f82420"
        },
        "ProductTitle": "name123",
        "ProductCost": 12.99
    },
    {
        "_id": {
            "$oid": "65fde45ec4b66f6ce05521e8"
        },
        "ProductTitle": "Product 1",
        "ProductCost": 10
    },
    {
        "_id": {
            "$oid": "65fde895dcbe6af2d64ff36a"
        },
        "ProductTitle": "Product 1",
        "ProductCost": 10
    },
    {
        "_id": {
            "$oid": "66152c18793dab197bd9c3c0"
        },
        "ProductTitle": "Product 2",
        "ProductCost": 15.99
    }
]
```

```
found keyword
{
    "productTitles": [
        "name1",
        "name123",
        "Product 1",
        "Product 1",
        "Product 2"
    ]
}
```

The root page expects to have each url for the other pages so this test checks for the piece of text /getProducts for this url

```
found keyword
{
    "/getProducts": "getting all products from db",
    "/insertProduct": "inserting products",
    "/getTitles": "titles of products"
}

found keyword
```

```python
# Test 1
name = 'Test 1'
url = 'http://localhost:5000/getProducts'
result = checkServiceForWord(url, 'name1')
saveResult(name, url, result)


# Test 2
name = 'Test 2'
url = 'http://localhost:5000/getTitles'
result = checkServiceForWord(url, 'name1')
saveResult(name, url, result)


#Test 4
name = 'Test 4'
url = 'http://localhost:5000/'
result = checkServiceForWord(url, '/getProducts')
saveResult(name, url, result)
```

## 4. Results from test results

The results from this test file are stored in a file on the Jenkins server. Below is the file. They are saved to a file called test_saved.log

```
test_saved.log
Test name:Test 1
Test URL:http://localhost:5000/getProducts
Test result:True
--------------------------------------------
 Test name:Test 2
Test URL:http://localhost:5000/getTitles
Test result:True
--------------------------------------------
 Test name:Test 4
Test URL:http://localhost:5000/
Test result:True
--------------------------------------------
```

Here is the function that creates the log file

```
f = open('test_saved.log', 'w+')
def saveResult(name, url, result):
    f.write('Test name:' + str(name) + '\n')
    f.write('Test URL:' + str(url) + '\n')
    f.write('Test result:' + str(result) + '\n')
    f.write('----------------------------------------------\n ')
```

It was a function given to use by the lecturer.

## 5. Stopping all servers started.

Here is the mongoDB docker container being stopped after all the unit tests are done being run from the test.py file.

```
C:\ProgramData\Jenkins\.jenkins\workspace\test2>docker-compose stop
 Container test2-mongo-1  Stopping
 Container test2-mongo-express-1  Stopping
 Container test2-mongo-express-1  Stopped
 Container test2-mongo-1  Stopped
```

## 6. Message of Jenkins

The message was just displayed using an echo in the jenkins bash console here is the screenshot below of it running

```
C:\ProgramData\Jenkins\.jenkins\workspace\test2>echo "Process completed and ready for
the customer"
"Process completed and ready for the customer"

C:\ProgramData\Jenkins\.jenkins\workspace\test2>exit 0
Finished: SUCCESS
```