CKCS 145: Lab 4 - Middleware with Backend

In this lab, you will be working with Python, Flask, MongoDB and the command prompt. You will need to use the virtual machine (VM) that was set up in Lab 1. To start your VM, log into the machine using your account ("TorontoMet" is the username and "torontomet123" is the password). Open up a Terminal or command prompt and start with step 1.

Prerequisites:

- Flask version 3.0.3 or higher
- Werkzeug 3.0.3 or higher
- MongoEngine 0.27.0 or higher

1. Before we get started with this lab, we need to use MongoDB and create a database.
2. MongoDB is a headless service that runs on your VM. You can use MongoDB Compass, a GUI application to create and populate databases.
3. In MongoDB Compass, you need to create a database named "CKCS145".
4. Inside the "CKCS145" database, you need to create a collection named "User".
5. To the collection named "User", we can add documents. Add the documents below. The names represent well known graduates of Ryerson University. The email addresses are not real email addresses, and have been made up for the purpose of this lab.

```
[
  { "name": "Gail Kim",
    "email": "gk@impactwrestling.com" },
  { "name": "Isadore Sharp",
    "email": "is@fourseasons.ca" },
  { "name": "Caitlin Cronenberg",
    "email": "cg@photos.ca" }
]
```

6. Now that you have data in MongoDB, you may create an application in order to modify and access it. Remember there is a password set on the MondoDB database. The MongoDB database may be accessed using port 27017.
7. Create a folder named "lab4". This folder can be in your home folder or in your project folder.

```
mkdir lab4
```

8. Change the current directory to "lab4". This can be achieved using the cd command.

```
cd lab4
```

9. Install the necessary tools to create a Flask application. From the command prompt, you can run the following commands.

```
pip install mongoengine
```

10. Create a text file named “lab4-app.py”.

```
touch lab4-app.py
```

11. Open the above text file using a text editor and add the following inside it.

```
from flask import Flask, jsonify, request, render_template


import mongoengine as db


import json


# Create Flask application object
app = Flask(__name__)


# Create connection to CKCS145 MongoDB database
client = db.connect('CKCS145', username='', password='')



# A test route to determine if Flask application is initialized
properly.
# http://localhost:5000/test
@app.route('/test', methods=['GET'])
def test_route() :
    return 'test'


if __name__ == '__main__':
    app.run(debug=True)
```

12. Start the above application.

```
python3 lab4-app.py
```

13. The above application is Flask middleware with a single route. This means that it is a service that may be accessed on port 5000 using a web browser. The URL that may be used is http://localhost:5000/test/. What do you see in your web browser when you visit this URL?
14. Create a data model to access the document in MongoDB. The code below can be added above the route decorator used in test_route() method in lab4-app.py file.

```
# Data Class for accessing MongoDB collection
class User(db.Document):
    name = db.StringField()
    email = db.StringField()
    meta = {'collection': 'User', 'allow_inheritance': False}
```

15. Create a route for retrieving all documents from MongoDB. These routes can be added after test_route() method in lab4-app.py file.

```
# A route to list all users.
# http://localhost:5000/user/list
@app.route('/user/list', methods = ['GET']) #Enable GET and POST
def list_all_users():
    data = list(User.objects)  # Create a list
    return json.loads( User.objects.to_json() )
```

    The above route can be tested by running the application. To run the application, use the command from step 10.

16. Open a web browser and enter in the url: http://localhost:5000/user/list. You should see a JSON list of users. These users are retrieved by the middleware from the database.
17. Create a route for inserting a new document into MongoDB.

```
# A route that allows the insertion for a single user.
# http://localhost:5000/user/insert
@app.route('/user/insert', methods=['POST'])
def create_user():
    name = request.form.get('user_name')
    email = request.form.get('user_email')
    # create a user object with above data
    new_user = User(name=name, email=email)
    # persist user object to database
    new_user.save()
```

```
        return json.loads( new_user.to_json() )

        return new_user.to_json()
```

18. The route in the above step requires a POST request from an html form to submit a name and email. RESTer can be used to submit such a request. Use figures 1 and 2 below to configure a POST request.

Figure 1: Configuration of content type for POST request.

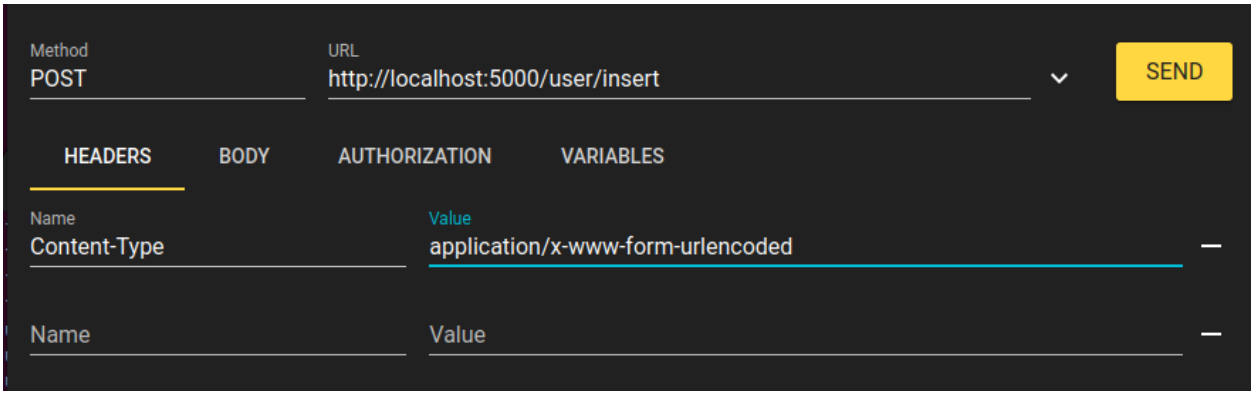
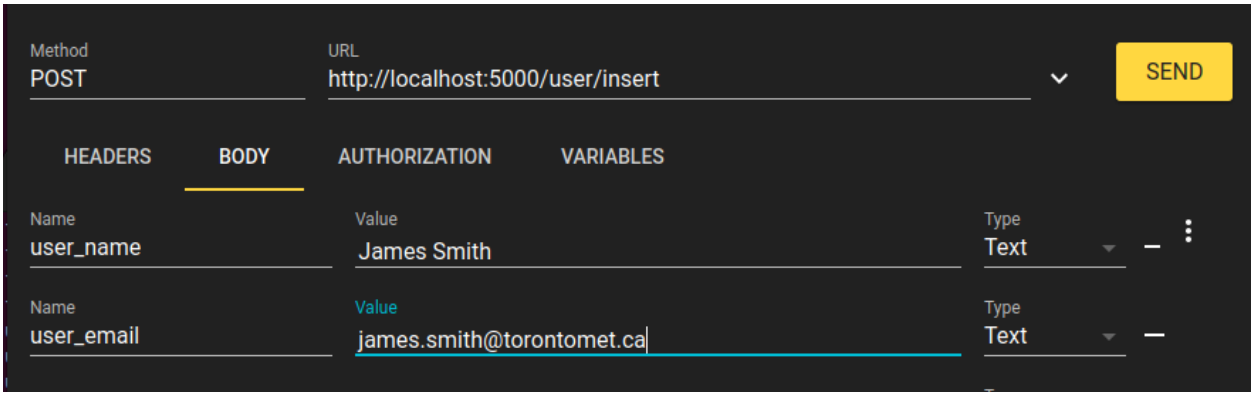


Figure 2: Configuration of “user_name” and “user_email” text fields and corresponding values.



19. Create a route for finding a User. To the route you must submit a user’s name.

```
# A route that allows to search for a single user in the database.
# http://localhost:5000/user/find
@app.route('/user/find', methods=['POST'])
def find_a_user():
    # retrieve data from html form
    name = request.form.get('user_name')
    # find all names that match
    users = User.objects(name=name)
    # get the first name from matching names
    user = users.first()

    if not user:
        # it is possible that no names were found
        return jsonify({'error': 'data not found'})
    else:
        # return user object that matches our search criteria
        return json.loads( user.to_json() )
```

20. Create a route for deleting a User. To the route you must submit a user’s name.

```
# A route that allows the deletion for a single user.
# http://localhost:5000/user/delete
@app.route('/user/delete', methods=['post'])
def delete_user():
```

```
# retrieve data from html form
name = request.form.get('user_name')
# find all names that match
users = User.objects(name=name)
# get the first name from matching names
user = users.first()

if not user:
    return jsonify({'error': 'data not found'})
else:
    user.delete()

status_message = user.name + ' has been deleted'
return jsonify({'status': status_message})
```

21. Use RESTer to create a URL POST request for the routes created in the above steps.
22. In this lab, only GET and POST requests are used. Why are PUT and DELETE requests not used?
23. If problems are encountered to complete this lab, it is recommended that debugging functionality be turned on. Stack traces can be viewed in the terminal where this app is running from. When errors are generated by web pages, stack traces can be used for identifying errors in your code. Alternatively a stack trace can be viewed using your browser using a debug PIN. By default debugging should be turned on.