Connecting to MongoDB from Flask

Estimated time needed: 30 minutes

In the last video, you learned how to connect and query MongoDB from Python. Since Flask is written in Python, it is easy to use MongoDB with Flask using the `PyMongo` module.

Objectives

In this reading, you will learn how to:

1. Initiate a PyMongo client in Flask
2. Connect with MongoDB database
3. Work with database and collections in MongoDB from Flask
4. Return appropriate HTTP status from Flask routes

Directory structure

Let's look at a sample directory structure to see how you can set up PyMongo to work with a database in Flask.

```
.
├── __init__.py
├── app.py
├── db.py
└── venv
```

1. `__ini__.py`: empty file that initializes a python module
2. `app.py`: main application that contains the Flask routes
3. `db.py`: python file to initialize the database
4. `venv`: python virtual environment

Initiate MongoDB client from Flask

In this exercise, you will initiate a MongoDB client. Assume you have MongoDB running on the localhost and can connect with the username, `mongouser`, and password, `password`. The following command will initiate a client using these credentials:

```
client = MongoClient('mongodb://%s:%s@127.0.0.1' % ('mongouser', 'password'))
```

Populate the database for the first time

You will create a `PyMongo` client in the `db.py` file. This can then be imported in the main app file.

```
from pymongo import MongoClient
```

Next, the following function initializes the `MongoClient` client and inserts three Todos in the `todo` collection that belongs to the `tododb` database.

```
def init_db():
    client = MongoClient('mongodb://%s:%s@127.0.0.1' % ('mongouser', 'password'))
    client.tododb.todo.drop()
    client.tododb.todo.insert_many(
        [
            {"priority": "high",
            "title": "Get milk"},
            {"priority": "medium",
            "title": "Get gasoline"},
            {"priority": "low",
            "title": "Water plants"}
        ]
    )
    return client
```

1. `Line1`: first, create a new PyMongo client using the URL of the form `mongodb://%s:%s@127.0.0.1`. You replace the username and password. This secret information should not be added to the code. In this learning activity, the information is included in the code for easy understanding.
2. `Line 2`: creates a connection to MongoDB, as explained above.
3. `Line 3`: drops the `todo` collection from the `tododb` database. This means that every time the Flask server is restarted, you will get a fresh copy of the database in this sample application.
4. `Line 4`: inserts three todos into the collection.

Interact with MongoDB database

You will now use the `db.py` file in the Flask application. Let's look at the code:

```
app = Flask(__name__)
from . import db
client = db.init_db()
```

1. `Line 1`: initiates a new Flask application.
2. `Line 3`: imports the db file from the current `.` module.
3. `Line 4`: initiates and populates the database using the `db.init_db()` method.

Return JSON data and HTTP status

We can now start writing some routes. First route is for the URL `/todos`, and it will simply return all the documents in the collection.

```
@app.route("/todos")
def index():
    result = client.tododb.todo.find({})
    return json_util.dumps(list(result)), 200
```

1. `Line 1`: declares the `/todos` URL using the `@app.route` decorator.
2. `Line 2`: creates the method for handling this route.
3. `Line 3`: uses the `find()` method in `PyMongo` to retrieve all the documents in the `todo` collection from the `tododb` database.
4. `Line 4`: converts the `bson` structure returned by month into `json` using the `json_util` module. This module provides methods to explicitly convert BSON objects to JSON and vice versa.
5. `Line 4`: returns the JSON list with an HTTP code of 200.

Let's look at the case where you need to return a different HTTP code. The route is for the URL `/todos/<priority>` and should return all todos with the given priority. The priority can be **low**, **medium**, or **high**. If the client sends a GET request to `/todos/high`, the method will return all items with a priority of `high`. If the client sends a priority other than `high`, `medium`, or `low`, the code will send an `HTTP 404 NOT FOUND` back.

```
@app.route("/todos/<priority>")
def get_by_priority_better(priority):
    result = client.tododb.todo.find({"priority": priority})
    result_list = list(result)
    if not result or len(result_list) < 1:
        return json_util.dumps(result_list), 404
    return json_util.dumps(result_list), 200
```

1. `Line 1`: declares the `/todos/<priority>` URL using the `@app.route` decorator.
2. `Line 2`: creates the method for handling this route. The method takes the priority as an input. Flask automatically extracts this from the URL provided in the decorator.
3. `Line 3`: uses the `find()` method similar to the code above, but filters for todos with a priority of `priority` variable passed in by the client in the URL itself.
4. `Line 4`: the `find` method returns a cursor. This line converts the cursor into a list.
5. `Line 5`: checks if the list is empty.If so, the code returns a 404 HTTP status with the empty list.
6. `Line 8`: returns the list back to the client with an HTTP status of 200. Similar to the code above, you use `json_util` module to convert the `bson` structure into `json`.

An important point to remember in the code above is that once the `cursor` object returned from the `find` method is converted to a Python `list`, the cursor becomes empty. You now have to work with the `list` object moving forward, as you don't access the items in the cursor anymore.

Now that you know how to interact with the `MongoDB` database in `Flask` using `PyMongo` for two endpoints, you can create `PUT`, `POST`, and `DELETE` endpoints in the capstone project on your own.

Testing

Let's test the above two endpoints with the curl command.

1. We can run the application using the following command:

   ```
   flask run --reload --debugger
   ```

2. Test the `/todos` endpoint:

   ```
   curl -i -w '\n' localhost:5000/todos
   ```

   The result will look like:

   ```
   HTTP/1.1 200 OK
   Server: Werkzeug/2.2.2 Python/3.7.16
   Date: Fri, 10 Feb 2023 04:18:31 GMT
   Content-Type: text/html; charset=utf-8
   Content-Length: 273
   Connection: close
   [{"_id": {"$oid": "63e5c594f32480ef62a901ab"}, "priority": "high", "title": "Get milk"}, {"_id": {"$oid": "63e5c594f32480ef62a901ac"}, "priority": "medium", "title": "Get gasoline"}, {"_id": {"$oid": "63
   ```

   Formatted JSON output:

   ```
   [
     {
       "_id": {
         "$oid": "63e5c88077d86b4ad7dfa072"
       },
       "priority": "high",
       "title": "Get milk"
     },
     {
       "_id": {
         "$oid": "63e5c88077d86b4ad7dfa073"
       },
       "priority": "medium",
       "title": "Get gasoline"
     },
     {
       "_id": {
         "$oid": "63e5c88077d86b4ad7dfa074"
       },
       "priority": "low",
       "title": "Water plants"
     }
   ]
   ```

3. Test the `/todos/high` endpoint:

```
curl -i -w '\n' localhost:5000/todos/high
```

The command will return:

```
HTTP/1.1 200 OK
Server: Werkzeug/2.2.2 Python/3.7.16
Date: Fri, 10 Feb 2023 04:19:19 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 88
Connection: close
[{"_id": {"$oid": "63e5c594f32480ef62a901ab"}, "priority": "high", "title": "Get milk"}]
```

Formatted JSON output:

```
[
  {
    "_id": {
      "$oid": "63e5c88077d86b4ad7dfa072"
    },
    "priority": "high",
    "title": "Get milk"
  }
]
```

4. Test the `/todos/low` endpoint:

```
    HTTP/1.1 200 OK
Server: Werkzeug/2.2.2 Python/3.7.16
Date: Fri, 10 Feb 2023 04:21:07 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 91
Connection: close
[{"_id": {"$oid": "63e5c62b3d484ba15fb5184e"}, "priority": "low", "title": "Water plants"}]
```

Formatted JSON output:

```
[
  {
    "_id": {
      "$oid": "63e5c88077d86b4ad7dfa074"
    },
    "priority": "low",
    "title": "Water plants"
  }
]
```

Next Steps

You now know how to use `PyMongo` in a `Flask` application to interact with `MongoDB` database server. You can now continue to the next module, where you will write the second service for the capstone project and use the skills you learned in this reading.

Author(s)

CF