Step by Step Guide to Create CRUD Application

UI < - > Python < -- > MongoDB

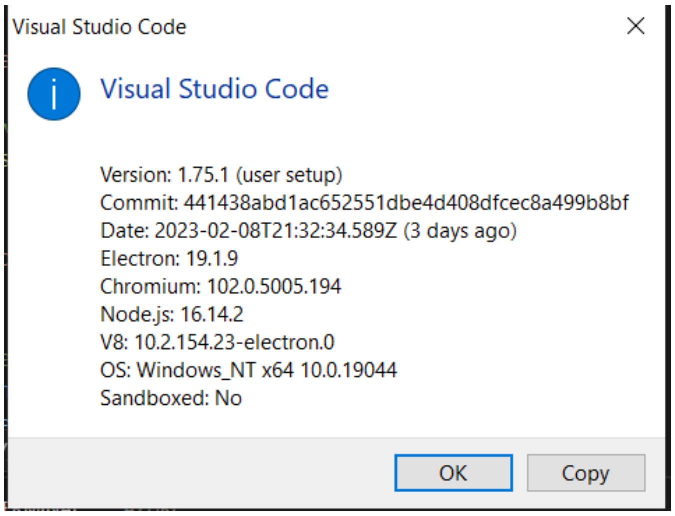
< - > Python < -- > SQLite DB

Created by: Bahadur Singh, 14 Feb 2023

This document will explain you steps about creating an application (MongoDB, SQLite, Python and HTML <py> scripting) technologies based application. You will learn the CRUD operations (Create, Read, Update and Delete) to both MongoDB (noSQL) and sqlite3 databases.

What is needed?

Visual Studio Code with below components.
Python Engine 3.9+ installed
MongoDB install on your local computer. You can also use cloud base ATLAS version.
Sqlite3 Database system on your system.
I am using this on Windows 10.



```
Version: 1.75.1 (user setup)
Commit: 441438abd1ac652551dbe4d408dfcec8a499b8bf
Date: 2023-02-08T21:32:34.589Z (3 days ago)
Electron: 19.1.9
Chromium: 102.0.5005.194
Node.js: 16.14.2
V8: 10.2.154.23-electron.0
OS: Windows_NT x64 10.0.19044
Sandboxed: No
```

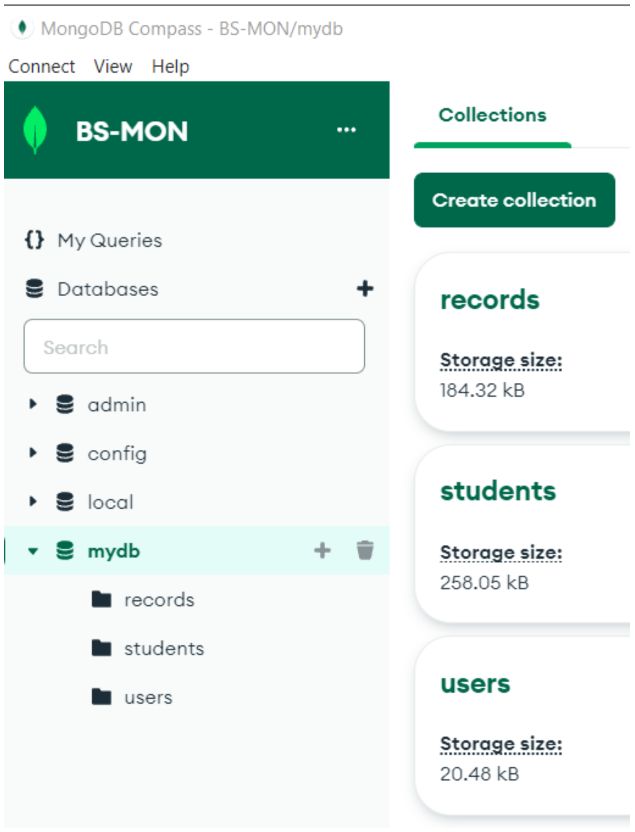
Command you needed to know on node JS Terminal.

```
>> pip install sqlite3
```

```
>> pip install pymongo
```

.. and other needed components.

MongoDB Setup



In this document, I will be using the collections under mydb (records or students)

Records have name, position, level, and Image fields. Image data is saved as base64 string.

MongoDB is noSQL Database. More documentation and details you can see mongodb.com

Collection **students** is used with this schema:

```
{
        name : string
        marks: number
        imagePath: string
        imgDataStr: string (base64 string)
}
```

sqlite3 Database

Install of sqlite3 is not part of this. Its an easy stuff to install Database.
Here the the schema about the data used.

Schema.sql

```
DROP TABLE IF EXISTS posts;

CREATE TABLE posts (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    created TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    title TEXT NOT NULL,
    content TEXT NOT NULL
);
```

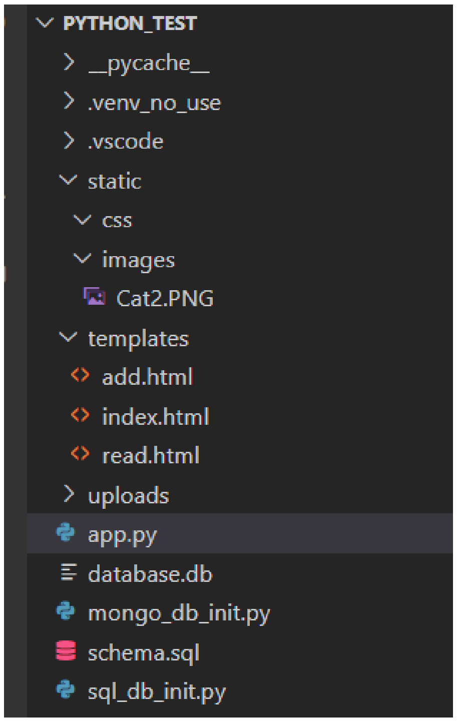
sql_db_init.py Create DB entries to start with experimenting.

```
import sqlite3
connection = sqlite3.connect('database.db')
with open('schema.sql') as f:
    connection.executescript(f.read())

cur = connection.cursor()
print ("DB Conncetion opened")
cur.execute("INSERT INTO posts (title, content) VALUES (?, ?)",
            ('First Post', 'Content for the first post')
            )
cur.execute("INSERT INTO posts (title, content) VALUES (?, ?)",
            ('Second Post', 'Content for the second post')
            )
cur.execute("INSERT INTO posts (title, content) VALUES (?, ?)",
            ('Third Post', 'Content for the third post')
            )
print ("INSERTS executed")
connection.commit()
connection.close()

print ("DB connection closed")
# Created by Bahadur Singh singh.bahadur@gmail.com
```

Server Side Application Structure:



```
PYTHON_TEST
  > __pycache__
  > .venv_no_use
  > .vscode
  v static
    v css
    v images
      Cat2.PNG
  v templates
    add.html
    index.html
    read.html
  > uploads
  app.py
  database.db
  mongo_db_init.py
  schema.sql
  sql_db_init.py
```

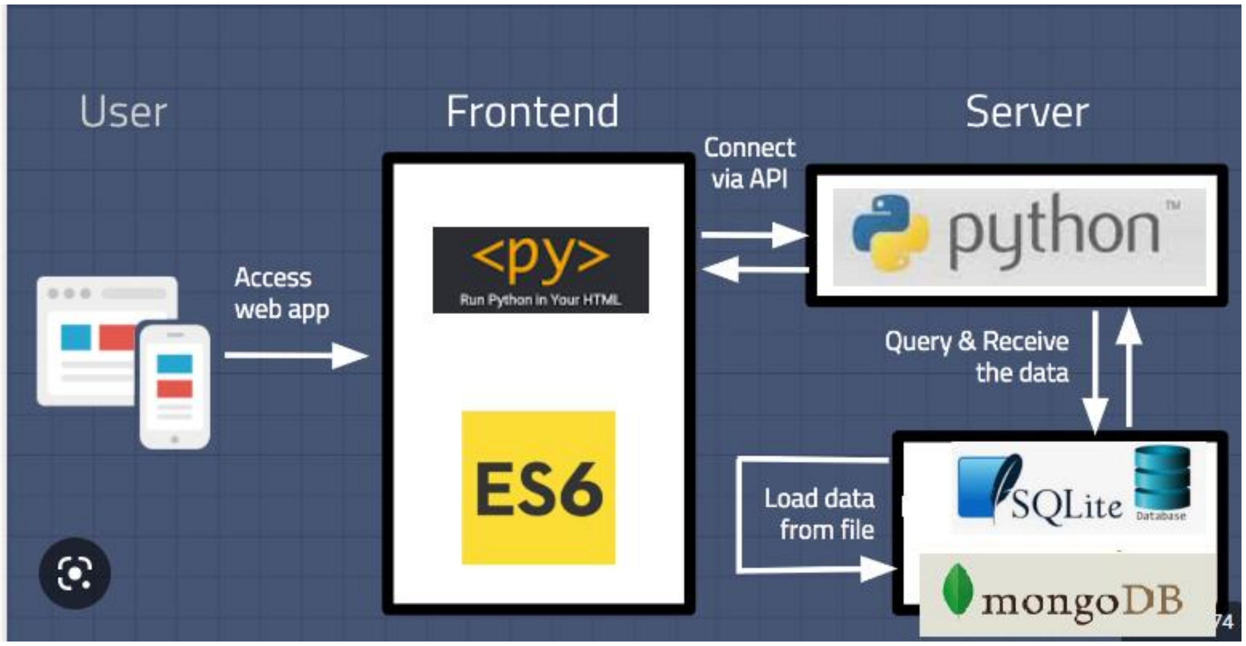
The project structure is defined like above image.

index.html, read.html, add.html	This file CLIENT CODE (FRONT-END), HTML rendering + django code with {{ expression }} notation, the list iteration for making
spp.py	Server side file manages application start and the name of URL access (called routes) and connection to databases
sql_db_init.py	Interface to sqlite database using sqlite3 library
mongo_db_init.py	Interface to Mongo DB using pymongi
database.db	Its database file
schema.sql	Schema file for SQLite DB

Client Side Application

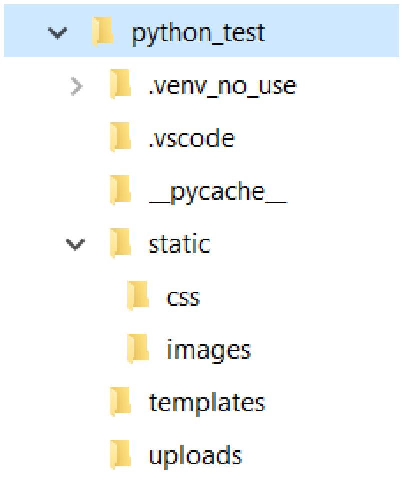
There is no client side code, its just an HTML and Django scripting {{ expression / data from server }} for dynamic data rendering.

Architecture Concept (Python connects to two Databases)



Code Explanation & Details

Lets understand first the server side. Goto Visual Studio Code, create a directory say python-test.



```
python_test
  .venv_no_use
  .vscode
  __pycache__
  static
    css
    images
  templates
  uploads
```

>> py app.py

>> flask run

One of these will start the server HTTP Listener at 5000 port (my setting)

Server Side Code and Implentation

app.py => main driver application file that starts the program at server on port 5000.

Now we will define the routes. The server API will open these functions. We will define routes functions that will expose middle-ware server API for the Client. Server application will be run on PORT 5000, it will connect to MongoDB & SQLite for DB CRUD operations.

HTML Client < --- localhost:5000 --- > Python Server API < --- -- > connects Databases

Routes we have defined for various client CRUD operation. Follow the code, its running example. There is also an independent upload file API as well. It copies the PNG file from client and put on the server. This uploaded file is attached as picture to the record.

/uploadFile route, you need to define for uploading the files. These files can be used as static link for reference on the web-page. My example shows an operation to write the PNG data to Database in base64 format and then read it back and render on the Browser.

```
import base64
import sqlite3, os
from flask import Flask , render_template, request, redirect, session
from werkzeug.utils import secure_filename

# call from different file function
from mongo_db_init import readMongoDBRecord, addMongoDBRecord, deleteMongoDBRecord,
updateMongoDBRecord

app = Flask(__name__)
print ("app created with Flask. ", __name__)

# File upload code from the browser
app.config['UPLOAD_FOLDER'] = 'uploads/'
app.config['ALLOWED_EXTENSIONS'] = set(['txt', 'pdf', 'png', 'jpg', 'jpeg', 'gif'])

def readMyName():
   print ('getMyName called')
   return

def get_db_connection():
   conn = sqlite3.connect('database.db')
   conn.row_factory = sqlite3.Row
   #print ("db_connection opened now ")
   return conn

def readDatabase():
   #print ('called readDatabase ??? ')
   conn = get_db_connection()
   global posts
   posts = conn.execute('SELECT * FROM posts').fetchall()
```

```
    #print ("app.py 2 db conn closed")
    conn.close()

    print ("app.py 2 def index SQL query done, Launch now index.xml >>>>> " )
    return render_template('index.html', posts=posts)
    #return posts

def readDatabase1():
    conn = get_db_connection()
    global posts
    posts = conn.execute('SELECT * FROM posts').fetchall()
    #print ("app.py 1  db conn closed")
    conn.close()

        print ("app.py 1 def index SQL query done, Launch now index.xml >>>>> " )
    #return render_template('index.html', posts=posts)
    return posts

def addRecordDatabase(tl, ct):
    conn = get_db_connection()
    conn.execute("INSERT INTO posts (title, content) VALUES ('" + tl + "', '" + ct +"')")
    print ("app.py db conn commit closed after INSERT ")
    conn.commit()
    conn.close()

    print ("app.py: INSERT DONE ==> Launch now index.xml >>>>> " )

def deleteRecordDatabase(inp):
    conn = get_db_connection()
    conn.execute("DELETE FROM posts WHERE TITLE ='" + inp + "'")
    print ("app.py db conn commit closed after delete ")
    conn.commit()
    conn.close()

    print ("app.py: DELETE DONE ==> Launch now index.xml >>>>> " )

# Set a secret key for encrypting session data
app.secret_key = 'my_secret_key'

# dictionary to store user and password
users = {
    'user1': '1',
    'user2': 'password2'
}

# The router is defined here
@app.route('/')
def gotoIndex ():
    return render_template('index.html')
```

```python
# calling read page using render_template
@app.route('/read')
def view_form():
    print ('view form called read MongoDB..')
    all_s = readMongoDBRecord ('Bahadur')
    all_p = readDatabase1 ()
    return render_template('read.html', all_students=all_s, posts=all_p)

#calling add.hml page
@app.route('/add')
def view_form1():
    # all_s = readMongoDBRecord ('Bahadur')
    all_p = readDatabase1 ()
    return render_template('add.html', posts=all_p)


# For handling get request form we can get
# the form inputs value by using args attribute.
# this values after submitting you will see in the urls.
# e.g http://127.0.0.1:5000/handle_get?username=xxx&password=yyyy223344
# this exploits our credentials so that's
# why developers prefer POST request.
@app.route('/handle_get', methods=['GET'])
def handle_get():
    if request.method == 'GET':
        #print ('got to GET')
        #username = request.args['username']
        actionType = request.args['actionType']
        print(" in GET " ,  actionType)
        return readDatabase ()
    else:
        return render_template('index.html')

# Data sent from the client is not exposed into the URL of POST request
@app.route('/handle_post', methods=['POST'])
def handle_post():
    if request.method == 'POST':
        title = request.form['title']
        content = request.form['content']
        actionType = request.form['actionType']

        print(' ==> actionType ' + actionType)
        if (actionType == 'addRecordSQL') : addRecordDatabase (title, content)
        if (actionType == 'deleteRecordSQL') : deleteRecordDatabase (title)
        # MongoDB functions
        if (actionType == 'deleteRecordMongoDB') : deleteMongoDBRecord (title)
        if (actionType == 'addRecordMongoDB') : addMongoDBRecord (title, content)
        return readDatabase()
    else:
        return render_template('index.html')

def allowed_file(filename):
    return '.' in filename and \
           filename.rsplit('.', 1)[1].lower() in app.config['ALLOWED_EXTENSIONS']

# Router upload for posting a file from browser. File saved local on server and can
# be uploaded to Mongo DB using pymongo library.

@app.route ('/upload', methods=['POST'])
def uploadFile ():
    #print ('>>> upload file started')
    name = request.form ['title']
    file = request.files['file']
    print ('>>> upload file started', file, name)
    #and allowed_file(file.name)
    if (file ):
        #print ('inside if')
        filename = secure_filename (file.filename)
        #print ('>>>>filename ' + file.filename)
        file.save (os.path.join (app.config['UPLOAD_FOLDER'], filename))
        print ('>>> file saved.', filename)

        img_file = open('./uploads/' + filename, "rb")
        my_string = base64.b64encode(img_file.read()).decode ('utf-8')
        #print(my_string)

        # my string is base64 data
        updateMongoDBRecord (name, my_string)
        return render_template('index.html',filename=filename)
    else:
        return gotoIndex

if __name__ == '__main__':
    app.run()

# Created by Bahadur Singh singh.bahadur@gmail.com
```

mongo_db_init.py => This is an interface to database object. You need MongoDB installed as an application. pymongo module install inside the node terminal console is needed using

```
>>  pip install pymongo
```

```
from pymongo import MongoClient
import pprint
```

```
def readMongoDBRecord (what):
    client = MongoClient ("mongodb://localhost:27017")
    print ('readRecord: MongoDB connected')
    db = client["mydb"] # database
    studentList = db.students # collection

    print ('readRecord: Get Collection: ', studentList)

    #one_student = studentList.find_one({"name":what}) # get recor
    #print (one_student['name'])

    all_students = studentList.find() # get all records
    print ('connection not closed ', all_students)
    return all_students

def addMongoDBRecord (name1, mark1):
    print ('add to MongoDB ')

    client = MongoClient ("mongodb://localhost:27017")
    db = client["mydb"] # database
    studentList = db.students # collection
    #print ('Get Collection: ', studentList)

    one_student = studentList.insert_one({"name":name1, "marks":mark1}) # get recor
    print ('record added')
    return

def deleteMongoDBRecord (name1):
    print ('delete from MongoDB ' + name1)

    client = MongoClient ("mongodb://localhost:27017")
    db = client["mydb"] # database
    studentList = db.students # collection
    #print ('Get Collection: ', studentList)

    one_student = studentList.delete_one({"name":name1}) # get recor
    print ('record deleted ', one_student)
    return

def updateMongoDBRecord (name1, imgDataStr1):
    print ('update MongoDB image for ' + name1)

    client = MongoClient ("mongodb://localhost:27017")
    db = client["mydb"] # database
    studentList = db.students # collection
    one_student = studentList.update_one({"name":name1}, {"$set" : {"imgDataStr":imgDataStr1}}) # set
record new value
    print ('record updated ', one_student.upserted_id)
    return

#readRecord ('SB')

# Created by Bahadur Singh singh.bahadur@gmail.com
```

Schema.sql => need to Database setup

```
DROP TABLE IF EXISTS posts;

CREATE TABLE posts (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    created TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    title TEXT NOT NULL,
    content TEXT NOT NULL
);
```

sql_db_init.py : connection to SQLite database access

```
import sqlite3
connection = sqlite3.connect('database.db')
with open('schema.sql') as f:
    connection.executescript(f.read())

cur = connection.cursor()
print ("DB Conncetion opened")
cur.execute("INSERT INTO posts (title, content) VALUES (?, ?)",
            ('First Post', 'Content for the first post')
            )
cur.execute("INSERT INTO posts (title, content) VALUES (?, ?)",
            ('Second Post', 'Content for the second post')
            )
cur.execute("INSERT INTO posts (title, content) VALUES (?, ?)",
            ('Third Post', 'Content for the third post')
            )
print ("INSERTS executed")
connection.commit()
connection.close()

print ("DB connection closed")
```

Client Side Code and Implementation.

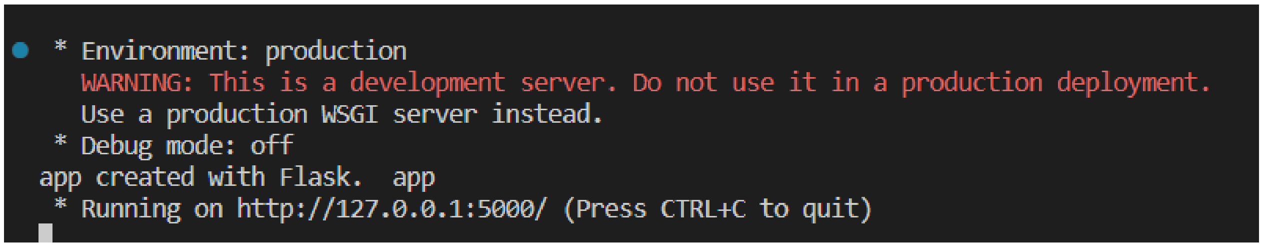
I used very small HTML client sample user-form.ejs file. It actually connects to server loacalhost:3001 < - - > MongoDB, You need to install MongoDB on your Window10 PC.

Copy-Paste code in your environment. Make sure MongoDB database setup installation runs on default localhost, it has no user password defined. Just an example to know CRUD functions.

To run this Program

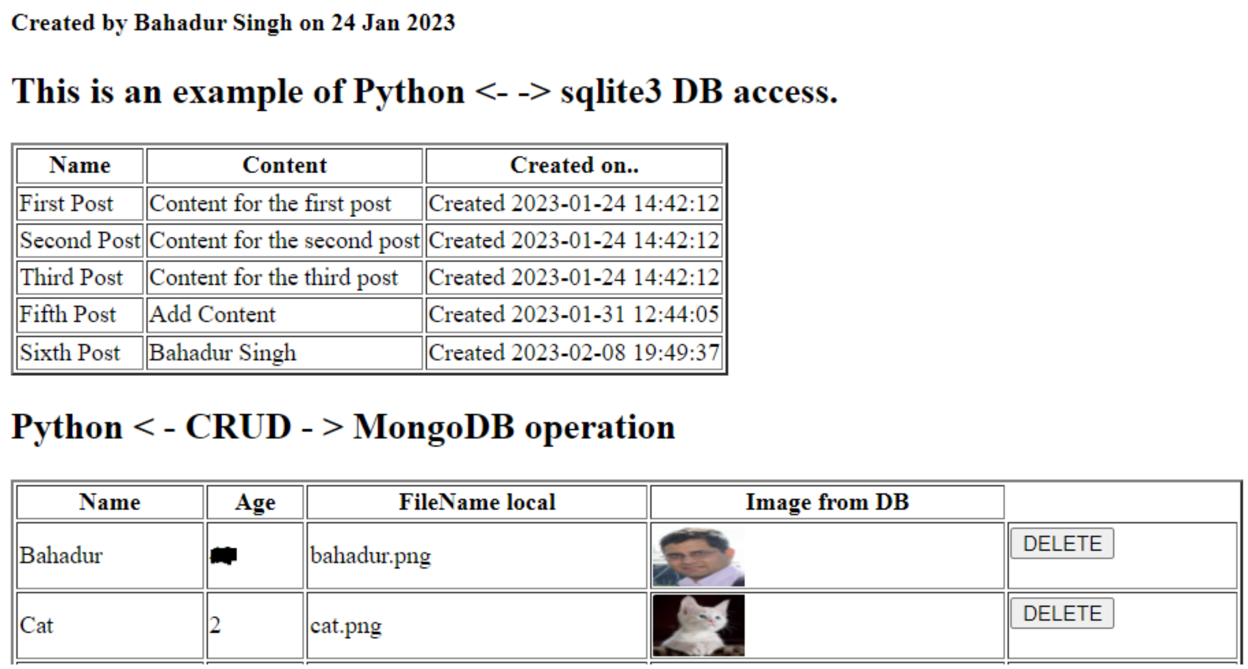
Start Server Side Terminal from below command

>> flask run



```
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
app created with Flask.  app
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Run this URL from your browser localhost://5000/



Created by Bahadur Singh on 24 Jan 2023

This is an example of Python <- -> sqlite3 DB access.

Name	Content	Created on..
First Post	Content for the first post	Created 2023-01-24 14:42:12
Second Post	Content for the second post	Created 2023-01-24 14:42:12
Third Post	Content for the third post	Created 2023-01-24 14:42:12
Fifth Post	Add Content	Created 2023-01-31 12:44:05
Sixth Post	Bahadur Singh	Created 2023-02-08 19:49:37

Python < - CRUD - > MongoDB operation

Name	Age	FileName local	Image from DB	
Bahadur	[redacted]	bahadur.png		DELETE
Cat	2	cat.png		DELETE

Index.html

```
<html>
<title> A Python Test App</title>
<body>

<h2>Created by Bahadur Singh on for Test (01-Feb-2023) </h2>

<hr>
<ul>
```

```
    <li>   <a href="http://localhost:5000/read"> Read Data both sqlite3 & MongoDB </a> </li>
    <li>   <a href="http://localhost:5000/add"> Add / Delete Record sqlite3 </a> </li>
</ul>
<hr>
File read ?? {{ filename}}
<pre>
    How to run it:
    Select Project -> app.py => Run in Terminal
    Terminal Commands to init sqlite3 DB:
        >> & c:/python39/python.exe sql_init_db.py
        >><b> flask run</b>
            This will open the HTTP server app at site port 5000
        It can now pass data to Server (Python) and can be checked on server side.
        localhost:5000/

        or
        >> py app.js will run same localhost:5000/

</pre>

</body>
</html>
```

Read.html

```
<html>
<title> Py Test App</title>
<body>

<h4>Created by Bahadur Singh on 24 Jan 2023</h4>

<h2> This is an example of Python <- -> sqlite3 DB access. </h2>

        <table border="2">
            <tr><th>Name</th> <th>Content</th><th>Created on..</th></tr>
            {% for post in posts %}
            <tr>
                <td>{{ post['title']}} </td><td>  {{ post['content']}} </td><td> Created {{ post['created'] }}</td>
                <!--
                <td>
                    <form name="{{ post['title']}}" method="POST" action="{{ url_for('handle_post') }}">
                        <input type="hidden" name="title" value="{{ post['title']}}"/>
                        <input type="hidden" name="content" value="{{ post['content']}}"/>
                        <button type="submit" name="actionType" value="deleteRecordSQL" >DELETE</button>
                    </form>
                </td> -->
            </tr>
        {% endfor %}
        </table>
```

```
    <div>
      <!-- url_for will route the forms request to
      appropriate function that user made to handle it.-->
      <!--we will retrive submitted values of inputs
      on the backend side using 'name' field of form.
          <h3>Handle GET Request DELETE action</h2>
          <form method="GET"
             action="{{ url_for('handle_get') }}">
            <input type="text" name="username" placeholder="Username">
            <button type="submit">DELETE</button>
          </form>
        </div>
        -->
```

```
        <div>
          <h2>Python < - CRUD - > MongoDB operation</h2>
          <table border="2" width="800">
            <tr><th>Name</th> <th>Age</th><th>FileName local</th> <th>Image from DB</th></tr>
            {% for stud in all_students  %}
              <tr>
                <td>{{ stud['name']}} </td><td>  {{ stud['marks']}} </td><td> {{ stud['imagePath'] }}</td>
                <td><img width=60 height=40 src="data:image/png;base64,{{ stud['imgDataStr']
}}"/></td>
                <td>
                  <form method="POST" action="{{ url_for('handle_post') }}">
                    <input type="hidden" name="title" value="{{ stud['name']}}"/>
                    <input type="hidden" name="content" value="xxx"/>
                    <button type="submit" name="actionType"
value="deleteRecordMongoDB">DELETE</button>
                  </form>
                </td>
              </tr>
                             {%  endfor %}
          </table>
        </div>

        <!--
        <img src="Cat2.png" /> not shown. I have not defined route to it
        PATH = {{ __dirname__ }}
        -->


<pre>
  Image static can be only loaded from static folder. css and others as well.
  html files should be in templates  url_for static/images/** picks for display
```

```
    <img src="{{ url_for('static', filename='images/cat2.png') }}" alt='missing image'/>
</pre>

<h4>This file should be in the ./templates/ folder. This is rendered usign the ==>
render_template('index.html') </h4>

</body>
</html>
```

Add.html (Add / Update / Delete operations)

```
<html>
<body>
    <h2>An example of Python <- CRUD -> sqlite3 DB access. </h2>
    <hr>
    <div>
    <!-- url_for will route the forms request to
    appropriate function that user made to handle it.-->
    <!--we will retrive submitted values of inputs
    on the backend side using 'name' field of form.-->
        <h2>ADD Record action SQL DB (title & content)</h2>
        <form method="POST"
                action="{{ url_for('handle_post') }}">
            <input type="text"  name="title" placeholder="title">
            <input type="text"  name="content"  placeholder="Content">
            <button type="submit" name="actionType" value="addRecordSQL">ADD RECORD</button>
        </form>
    </div>

       <table border="2">
          <tr><th>Name</th> <th>Age</th><th>Created</th></tr>
          {% for post in posts %}
          <tr>
             <td>{{ post['title']}} </td><td>  {{ post['content']}} </td><td> Created {{ post['created'] }}</td>
             <td>
                <form method="POST" action="{{ url_for('handle_post') }}">
                   <input type="hidden" name="title" placeholder="title" value="{{ post['title']}}">
                   <input type="hidden" name="content" value="xxxx">
                   <button type="submit" name="actionType" value="deleteRecordSQL">DELETE</button>
                </form>
             </td>
          </tr>
           {% endfor %}
       </table>

    <hr>
    <div>
        <!-- url_for will route the forms request to
        appropriate function that user made to handle it.-->
        <!--we will retrive submitted values of inputs
        on the backend side using 'name' field of form.-->
        <h2>ADD Record action MongoDB (Name & Age)</h2>
        <form method="POST"
                action="{{ url_for('handle_post') }}">
            <input type="text"  name="title" placeholder="name">
            <input type="text"  name="content"  placeholder="age">
            <button type="submit" name="actionType" value="addRecordMongoDB">ADD RECORD</button>
        </form>

        <h2>Upload a file to server </h2>
        <form method="POST"  action="{{ url_for('uploadFile') }}" enctype="multipart/form-data" >
            <input type="text"  name="title" placeholder="name">
            <input type="file"  name="file"  placeholder="select file">
            <button type="submit" name="actionType" value="addRecordMongoDB">Update Image</button>
        </form>

    </div>

</body>
</html>
```

Created by Bahadur Singh: singh.bahadur@gmail.com works at NCR Corp as

SME / Project Manager. Leads the Software Development for the customer based projects.