Secure SQL Handling with Python

introduction

When I think about `SQL` first thing I imagen, random hacker trying to inject queries in my web site, for example: trying to access some data, editing my database. in the last lab I took SQL injection as concept, and I understand the hackers methods, so I have to do two things:

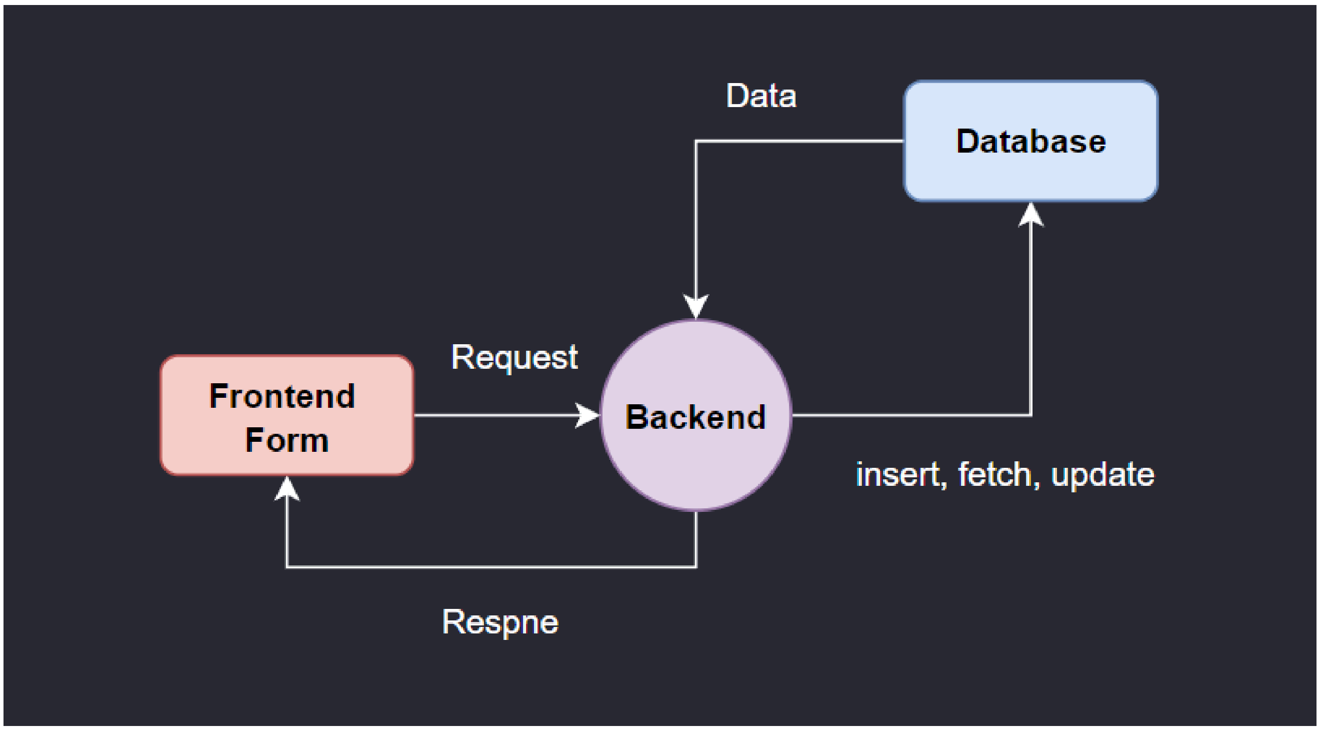
1. Python script that connect to a database.
2. Python backend that filter methods of SQL injection.

First thing: I have small knowledge about Data Bases ,but in general this is my first time with SQL commands, I used before non-relational-database.

Second thing: Most of students will use `sqllite3` because that's what copilot said, for me I will understand the pure code, and take this homework as project to learn new things, not only to pass the homework.

Project Details

When we need data from the database, or we want insert new data, we send request from frontend, and the backend handle it. then the backend has the right to do the changes in the database or send data.



Tip

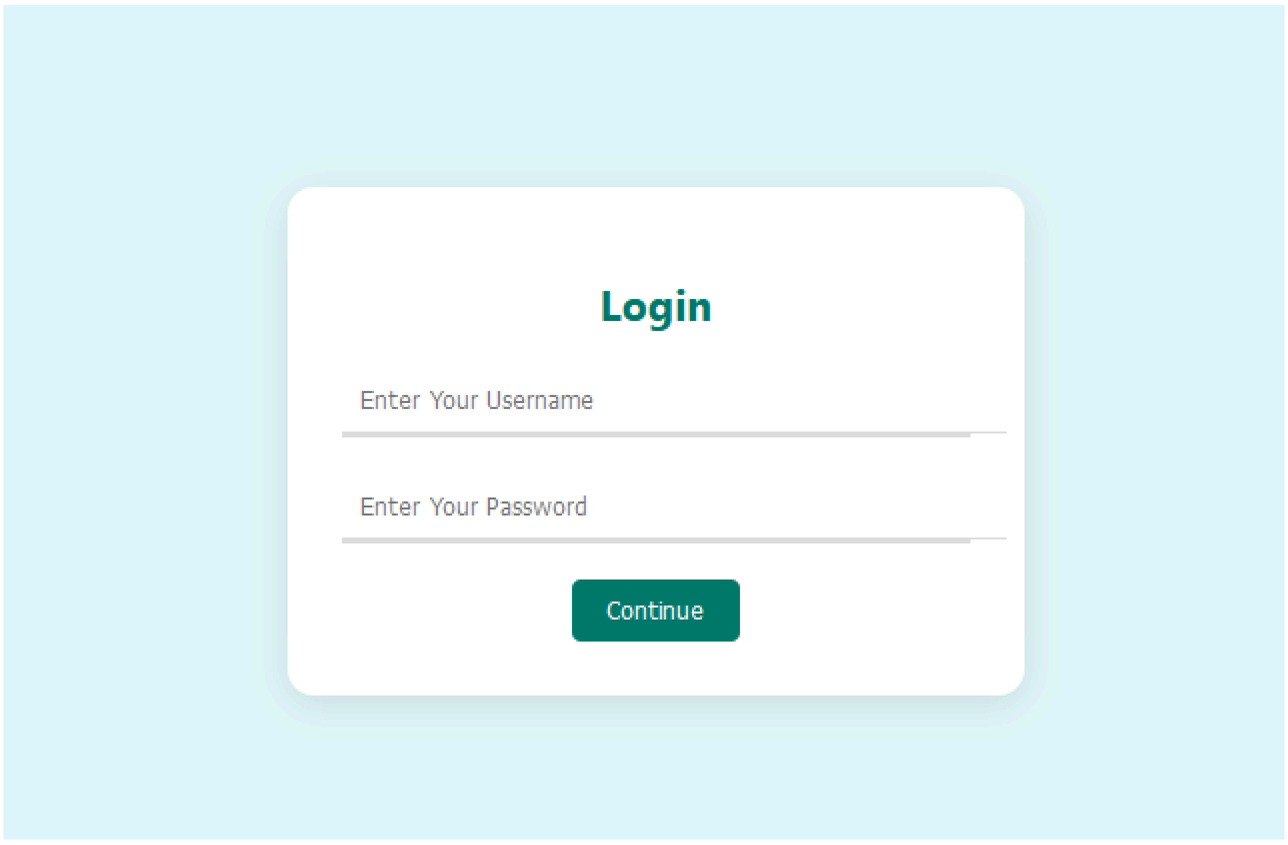
This flowchart is basic, because we could use an `API` to filter any `SQL injection` and we call it a `Secure API`

So the backend is the responsible for select and insert, for that we want a filter in python to prevent SQL injection

1. Frontend: HTML and CSS and JavaScript
2. Backend: Python
3. DataBase: Microsoft SQL Server

Frontend

I would not talk a lot about frontend files, you can check them but here is the login-form:



I didn't use any frameworks, I just used HTML and CSS and JavaScript

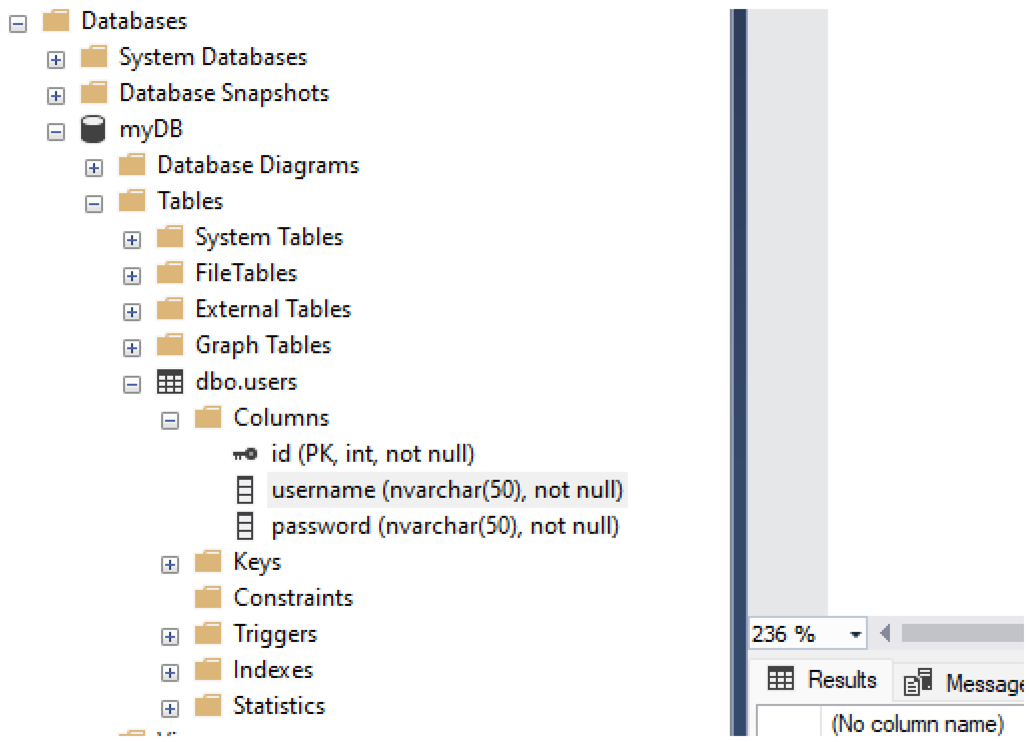
You can find the code of these files at `/Project-Files/template` and `/Project-Files/static` and it's standard way to organize files while building basic web pages like that

Database

I opened Microsoft SQL Server and then made a new database called `myDB` then I made a new query that create `users` table :

```
    CREATE TABLE users (
        id INT PRIMARY KEY IDENTITY(1,1),
        username NVARCHAR(50) NOT NULL,
        password NVARCHAR(50) NOT NULL
    );

    -- Insert example data
  INSERT INTO users (username, password) VALUES ('admin', 'admin');
```

So now we have the database:



```
Databases
  System Databases
  Database Snapshots
  myDB
    Database Diagrams
    Tables
      System Tables
      FileTables
      External Tables
      Graph Tables
      dbo.users
        Columns
          id (PK, int, not null)
          username (nvarchar(50), not null)
          password (nvarchar(50), not null)
        Keys
        Constraints
        Triggers
        Indexes
        Statistics
```

Backend

first: I should install `pyodbc` by running:

```
pip install pyodbc
```

it's library for connecting to Microsoft SQL Server and then I installed `Flask` to handle requests from user by running: `pip install Flask`

Code Connection

First of all, when we want to connect to a database, we want server name and database name and driver name , we put all that in string and send it using `pyodbc` library

```
import pyodbc as odbc
Connection = (
    r'DRIVER={SQL Server};'
    r'SERVER=X\SQLEXPRESS;'
    'DATABASE=myDB;'
    'Trusted_Connection=yes;'
)
Conn = odbc.connect( Connection )
```

in my case: my server name was `X\SQLEXPRESS` and I got the name using the command `SELECT @@SERVERNAME` in Microsoft SQL Server , and the `r' '` for the `\` in my server name, I got so many errors before noticing that.

Flask Code

Now after connecting to the database, we can filter username and password, and done the backend.

Let's import some modules:

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

1. `flask` is the module we use for making easy requests
2. `render_template` is a function provided by `Flask` that renders an HTML template and returns it as a response to the client.
3. `request` is an object provided by Flask that contains all the data sent by the client in an HTTP request
4. `re` is a module for working with regular expressions, we will use it for filtering the user input.

Tip

We could use `Flask-login` to make it easy, but in this project I didn't use it.

Example about regular expression module

```
import re
pattern = r'^[a-zA-Z0-9_]+$'
string = 'username123'
if re.match(pattern, string):
        print('Valid username')
```

the function `match()` will match the same pattern, and in this case the output is: **Valid username**, so that will help us for secure `SQL` handling like that:

```
    if not re.match(r'^[a-zA-Z0-9_]+$', username):
        return 'Invalid username format'
    if not re.match(r'^[a-zA-Z0-9_]+$', password):
        return 'Invalid password format'
```

here we say that username and password should be in regular expression, and that's the key for the whole project and I will use it later in my code.

Example about Flask

```
from flask import Flask

app = Flask (__name__)

@app.route('/')

def message():
    return 'hello there!'

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

So this a standard `Flask` app, `__name__` here is the name of the module, and it's a keyword that exist in every python module.

Then we route the user to `/` and make a default page that contain a **'hello there'** and in the end we run the flask app, and here the `__main__` is the python file.

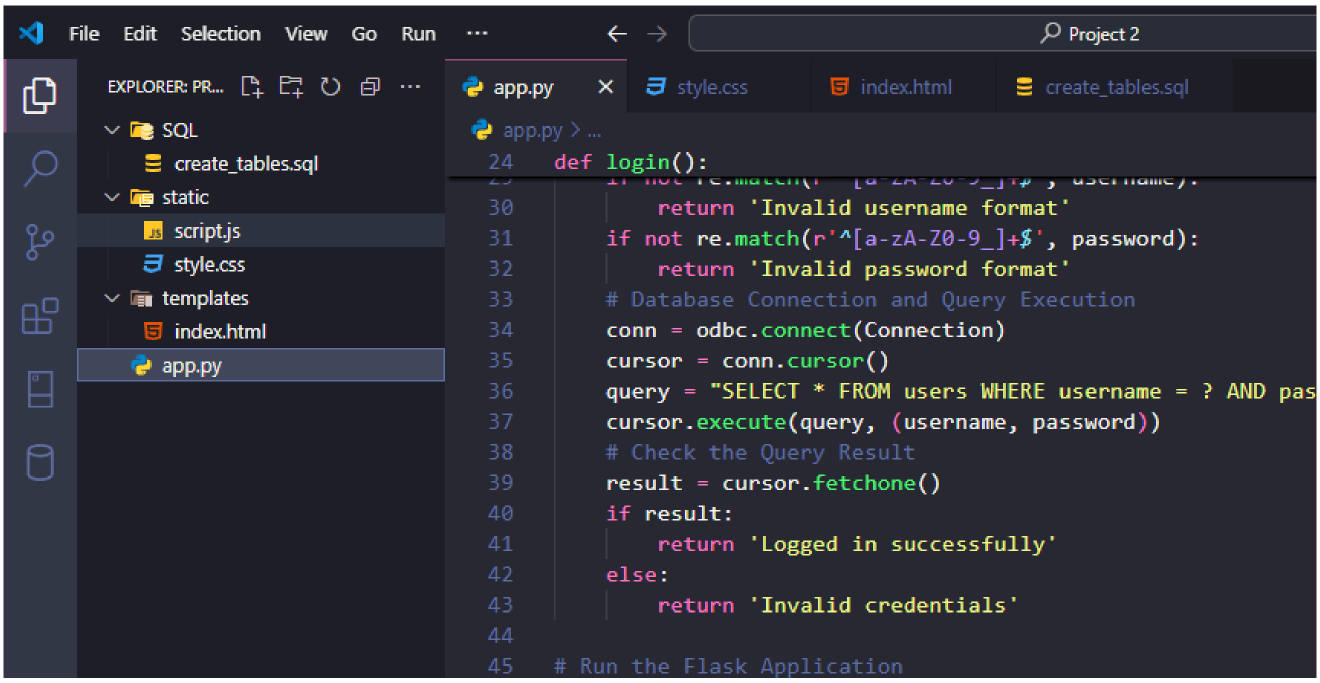
from these two examples, we can now build the idea we want, and we know python is so easy especially when we use modules .

Python full code

```
from flask import Flask, render_template, request

import re

app = Flask(__name__)

@app.route('/')

def home():
    return render_template('index.html')

@app.route('/login', methods=['POST'])

def login():
    username = request.form['username']
    password = request.form['password']

    # Validate input using regex to prevent SQL injection
    if not re.match(r'^[a-zA-Z0-9_]+$', username):
        return 'Invalid username format'

    if not re.match(r'^[a-zA-Z0-9_]+$', password):
        return 'Invalid password format'

    conn = odbc.connect(Connection)

    cursor = conn.cursor()

    query = "SELECT * FROM users WHERE username = ? AND password = ?"

    cursor.execute(query, (username, password))

    result = cursor.fetchone()

    if result:
        return 'Logged in successfully'

    else:
        return 'Invalid credentials'

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

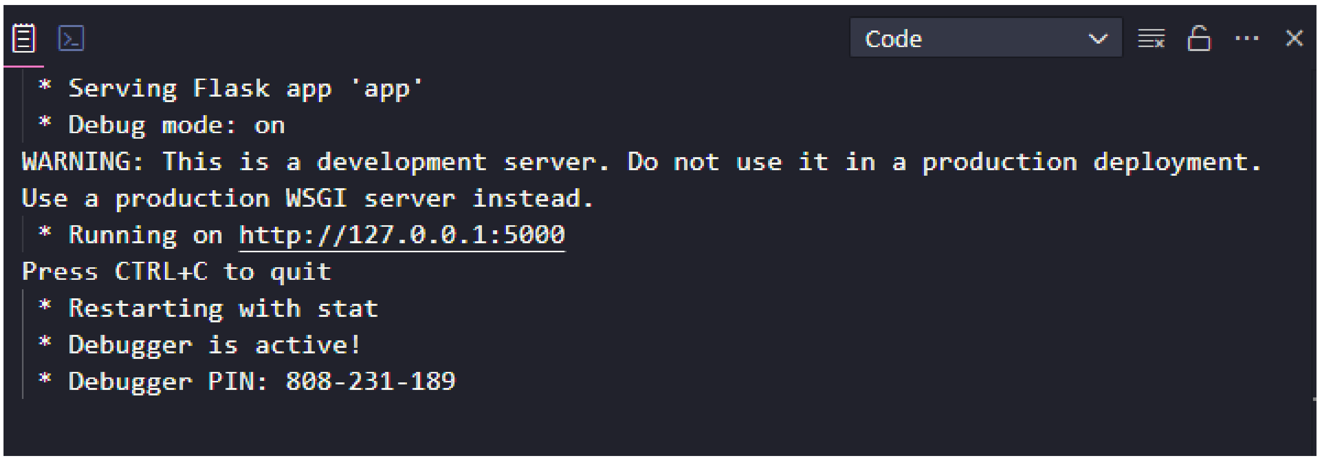
The Project

We are done! here are the project files in `/Project-Files` I used `Vs Code` to make this project. This also my first try with building `backend` and linking database .



```python
def login():
        return 'Invalid username format'
    if not re.match(r'^[a-zA-Z0-9_]+$', password):
        return 'Invalid password format'
    # Database Connection and Query Execution
    conn = odbc.connect(Connection)
    cursor = conn.cursor()
    query = "SELECT * FROM users WHERE username = ? AND pas
    cursor.execute(query, (username, password))
    # Check the Query Result
    result = cursor.fetchone()
    if result:
        return 'Logged in successfully'
    else:
        return 'Invalid credentials'

# Run the Flask Application
```

we run the code by: `python app.py` and here the output:

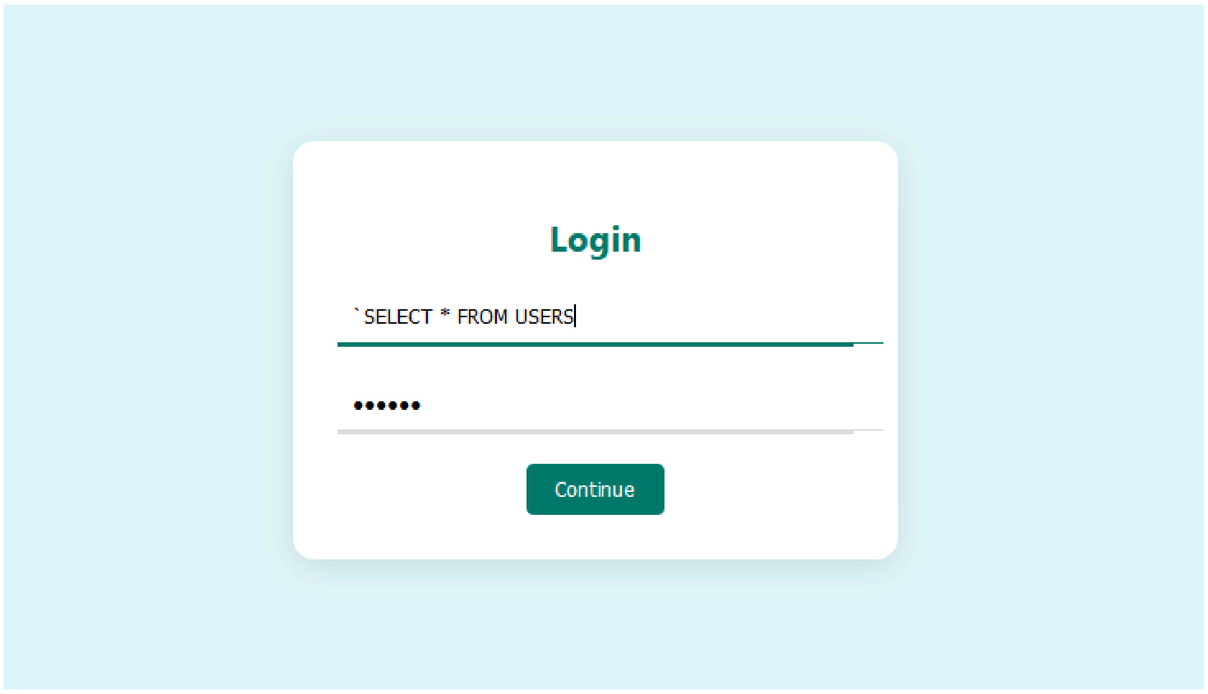


```
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 808-231-189
```

Everything is great! The flask app is running without errors, and this app won't end until I press `Ctrl + C` as developer, also as we can see: `Debug mode: on` and that's a choice I made it earlier when I faced a lot of errors.

> ✓ **State**
>
> App is running!

We can go to http://127.0.0.1:5000 for testing our secure backend, now I will try to inject SQL in the login-form:



But that or any other method won't work at all, and the response will be: `Invalid username format` or `Invalid password format` because only 4 lines of code:

```
    if not re.match(r'^[a-zA-Z0-9_]+$', username):
        return 'Invalid username format'
    if not re.match(r'^[a-zA-Z0-9_]+$', password):
        return 'Invalid password format'
```

That's it!

I also insert `/SQL/create_tables.sql` in the project files, I wanted at first to make everything easy, but I faced a lot of errors about `users` table is already exist, so I didn't use it in the python code, but I could use it by:

```
def execute_sql_file(file_path):
    with open(file_path, 'r') as file:
        sql_commands = file.read().split(';')
    conn = pyodbc.connect(conn_str)
    cursor = conn.cursor()
    for command in sql_commands:
        if command.strip():
            cursor.execute(command)
    conn.commit()
    cursor.close()
    conn.close()
```

This Project by: Ibrahem Hasaki.