Syrian Arab Republic

Lattakia - Tishreen University

Department of Communication and electrical engineering

5th , Network Programming : Homework No1

الجمهورية العربية السورية

اللاذقية ـجامعــة تشريـــن

كلية الهندسة الكهربائية والميكانيكية

قسم هندسة الاتصالات والالكترونيات

السنة الخامسة: وظيفة 1 برمجة شبكات

Question 1: Bank ATM Application with TCP Server/Client and Multi-threading

Project Description:

Build a TCP server and client Bank ATM application using Python. The server should handle multiple client connections simultaneously using multi-threading. The application should allow clients to connect, perform banking operations (such as check balance, deposit, and withdraw), and receive their updated account status upon completion.

Requirements:

A. The server should be able to handle multiple client connections concurrently.
B. The server should maintain a set of pre-defined bank accounts with balances.
C. Each client should connect to the server and authenticate with their account details.
D. Clients should be able to perform banking operations: check balance, deposit money, and withdraw money.
E. The server should keep track of the account balances for each client.
F. At the end of the session, the server should send the final account balance to each client.

Guidelines:

☺ Use Python's socket module without third-party packages.
☺ Implement multi-threading to handle multiple client connections concurrently.
☺ Store the account details and balances on the server side.

Notes:

☝ Write a brief report describing the design choices you made and any challenges faced during implementation.
☝ You can choose to create a TCP Server/Client Bank ATM application or any other appropriate application that fulfills all requirements.

Solution:

First, we set up the server using Python socket module to create a TCP server and this server will listen for the incoming connections on a specified port.

Next, we'll use the threading module to handle multiple client connections simultaneously so each client connection will be managed in a separate thread.

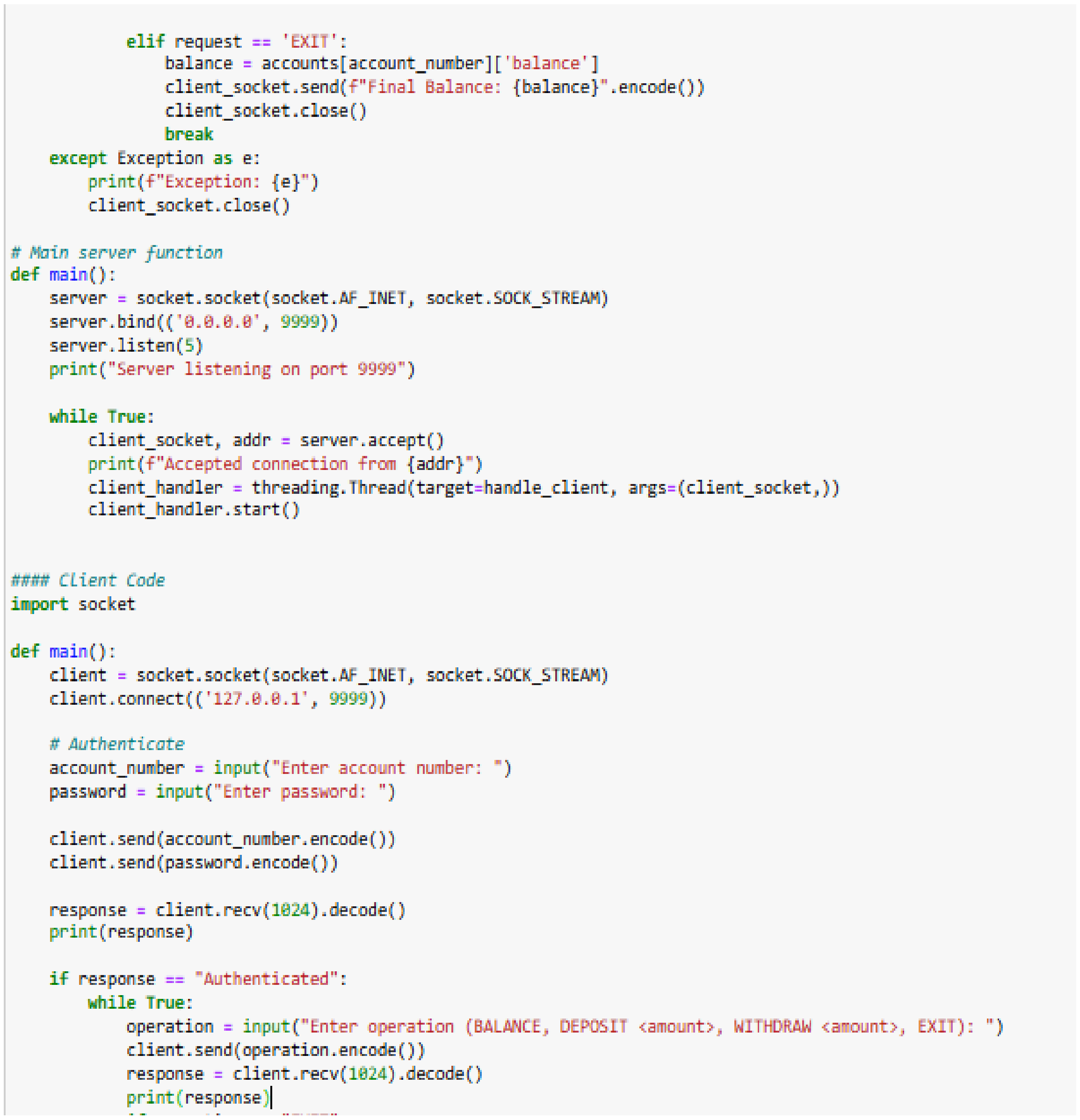
Next: We'll deal with the bank account and Authentication, we'll maintain a dictionary of predefined bank accounts and their balances, and the clients will authenticate using their account number and a password.

The clients can perform operations like check balance, deposit, and withdraw and the server will update account balances accordingly, then the server will send the final account balance to the client at the end of the session.

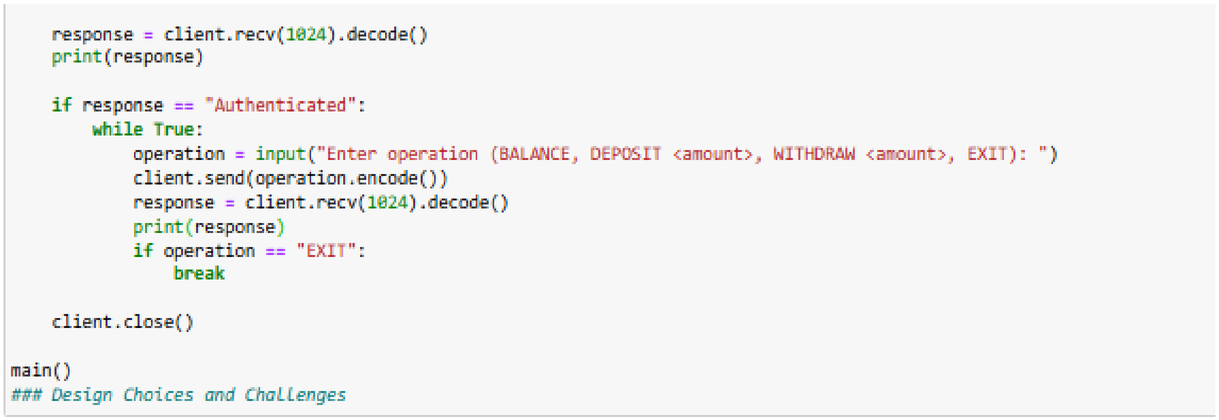
Next step: we connect the client to the server using the socket module by the step of the authentication when the client send the account details for authentication he's gonna connect to the server immediately , then he can perform the operation that he wants on the account, such as requests to check balance, deposit money, and withdraw money and so the client will receive responses from the server for each operation.

The code below showing the importing of the socket and threading and the predefined bank accounts with the authentication information:

In [ ]:
```python
#### Server Code
#python
import socket
import threading

# Predefined bank accounts
accounts = {
    '12345': {'password': 'pass1', 'balance': 1000},
    '67890': {'password': 'pass2', 'balance': 1500},
    '11223': {'password': 'pass3', 'balance': 1200},
}

# Function to handle client connection
def handle_client(client_socket):
    try:
        # Authenticate client
        account_number = client_socket.recv(1024).decode()
        password = client_socket.recv(1024).decode()

        if account_number in accounts and accounts[account_number]['password'] == password:
            client_socket.send("Authenticated".encode())
        else:
            client_socket.send("Authentication Failed".encode())
            client_socket.close()
            return

        # Handle client requests
        while True:
            request = client_socket.recv(1024).decode()

            if request == 'BALANCE':
                balance = accounts[account_number]['balance']
                client_socket.send(f"Balance: {balance}".encode())

            elif request.startswith('DEPOSIT'):
                amount = int(request.split()[1])
                accounts[account_number]['balance'] += amount
                client_socket.send(f"Deposited: {amount}".encode())

            elif request.startswith('WITHDRAW'):
                amount = int(request.split()[1])
                if accounts[account_number]['balance'] >= amount:
                    accounts[account_number]['balance'] -= amount
                    client_socket.send(f"Withdrawn: {amount}".encode())
                else:
                    client_socket.send("Insufficient funds".encode())
```



```
            elif request == 'EXIT':
                balance = accounts[account_number]['balance']
                client_socket.send(f"Final Balance: {balance}".encode())
                client_socket.close()
                break
    except Exception as e:
        print(f"Exception: {e}")
        client_socket.close()

# Main server function
def main():
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind(('0.0.0.0', 9999))
    server.listen(5)
    print("Server listening on port 9999")

    while True:
        client_socket, addr = server.accept()
        print(f"Accepted connection from {addr}")
        client_handler = threading.Thread(target=handle_client, args=(client_socket,))
        client_handler.start()


#### Client Code
import socket

def main():
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect(('127.0.0.1', 9999))

    # Authenticate
    account_number = input("Enter account number: ")
    password = input("Enter password: ")

    client.send(account_number.encode())
    client.send(password.encode())

    response = client.recv(1024).decode()
    print(response)

    if response == "Authenticated":
        while True:
            operation = input("Enter operation (BALANCE, DEPOSIT <amount>, WITHDRAW <amount>, EXIT): ")
            client.send(operation.encode())
            response = client.recv(1024).decode()
            print(response)
```



```
    response = client.recv(1024).decode()
    print(response)

    if response == "Authenticated":
        while True:
            operation = input("Enter operation (BALANCE, DEPOSIT <amount>, WITHDRAW <amount>, EXIT): ")
            client.send(operation.encode())
            response = client.recv(1024).decode()
            print(response)
            if operation == "EXIT":
                break

    client.close()

main()
### Design Choices and Challenges
```

Finally: we implement the code with a simple account number and password system for authentication, and we use the operations allowed to us to test the functionality.

Question 2: Simple Website Project with Python Flask Framework (you have choice to use Django or any Other Deferent Useful Python Project "from provide Project Links")

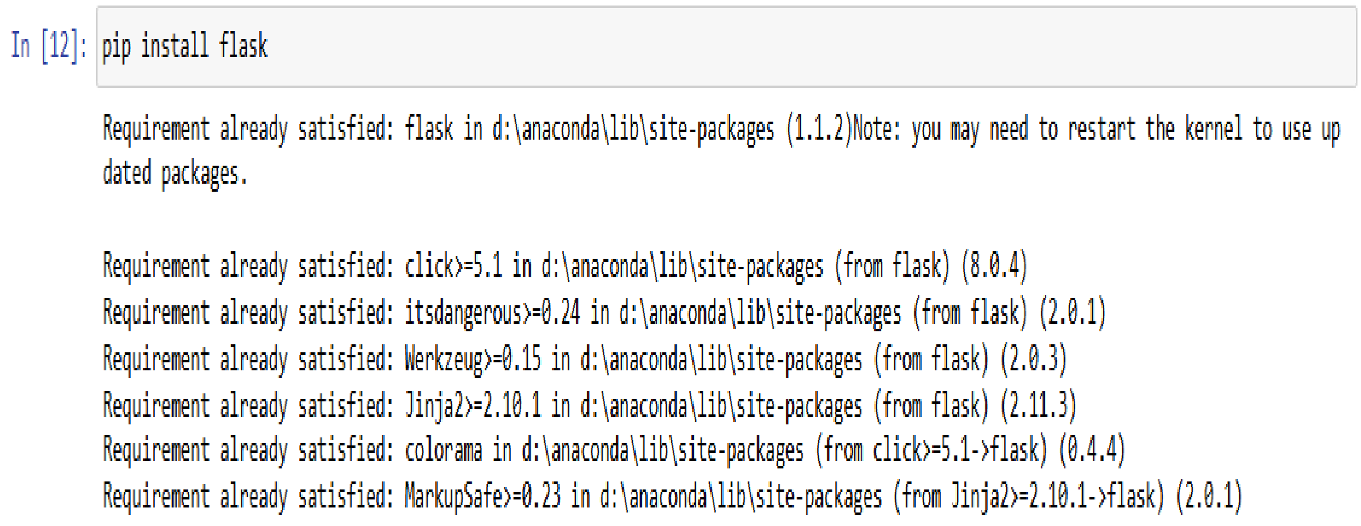
Create a simple website with multiple pages using Flask, HTML, CSS, and Bootstrap. The website should demonstrate your understanding of web design principles .

Requirements :

- **G.** Set up a local web server using XAMPP, IIS, or Python's built-in server (using Flask) .
- **H.** Apply CSS and Bootstrap to style the website and make it visually appealing.
- **I.** I. Ensure that the website is responsive and displays correctly on different screen sizes
- **J.** Implement basic server-side functionality using Flask to handle website features.

Solution:

First we create Flask App after installing the Flask using this instruction (pip install Flask) as shown below:

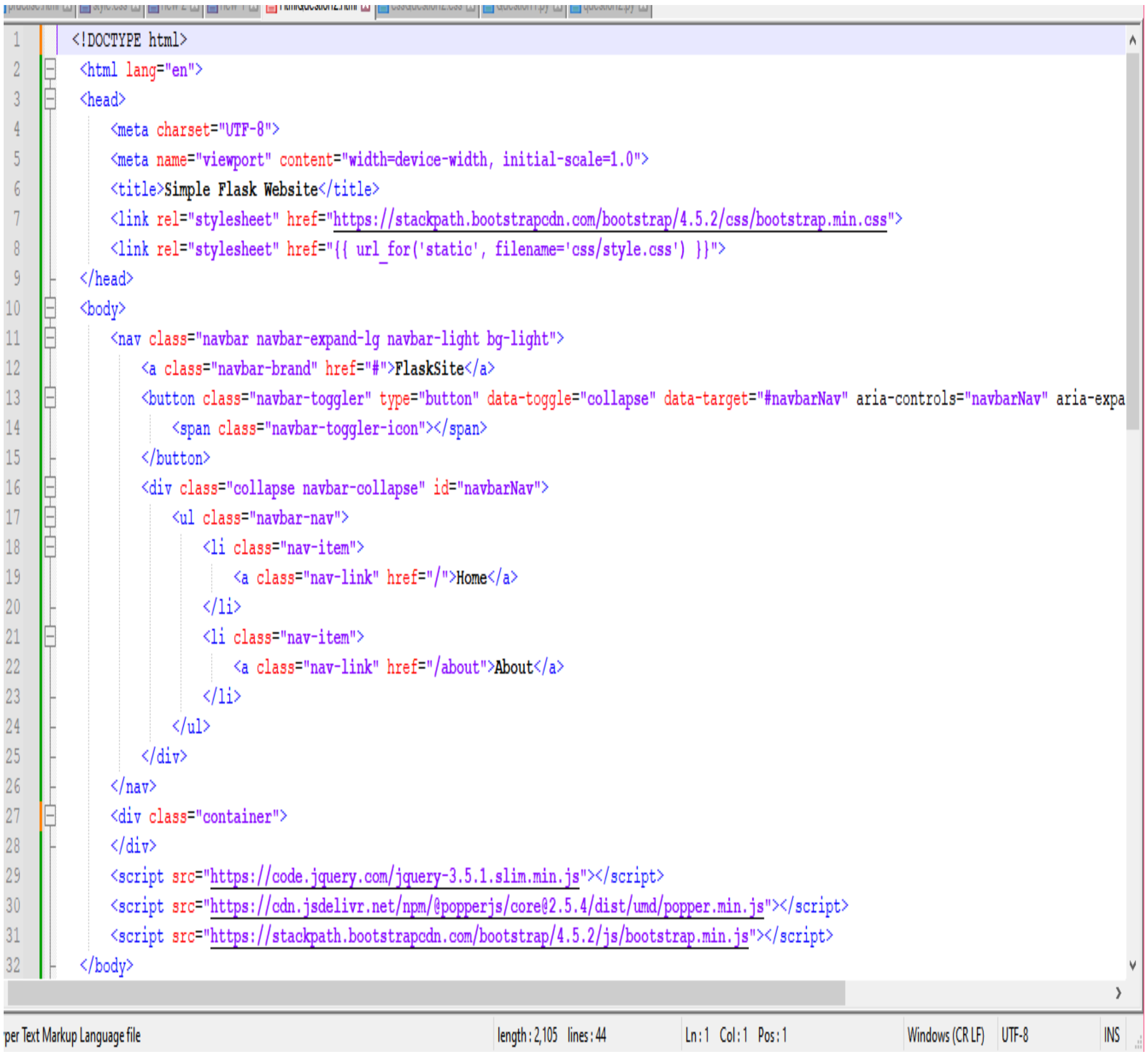


```
In [12]: pip install flask

Requirement already satisfied: flask in d:\anaconda\lib\site-packages (1.1.2)Note: you may need to restart the kernel to use up
dated packages.

Requirement already satisfied: click>=5.1 in d:\anaconda\lib\site-packages (from flask) (8.0.4)
Requirement already satisfied: itsdangerous>=0.24 in d:\anaconda\lib\site-packages (from flask) (2.0.1)
Requirement already satisfied: Werkzeug>=0.15 in d:\anaconda\lib\site-packages (from flask) (2.0.3)
Requirement already satisfied: Jinja2>=2.10.1 in d:\anaconda\lib\site-packages (from flask) (2.11.3)
Requirement already satisfied: colorama in d:\anaconda\lib\site-packages (from click>=5.1->flask) (0.4.4)
Requirement already satisfied: MarkupSafe>=0.23 in d:\anaconda\lib\site-packages (from Jinja2>=2.10.1->flask) (2.0.1)
```

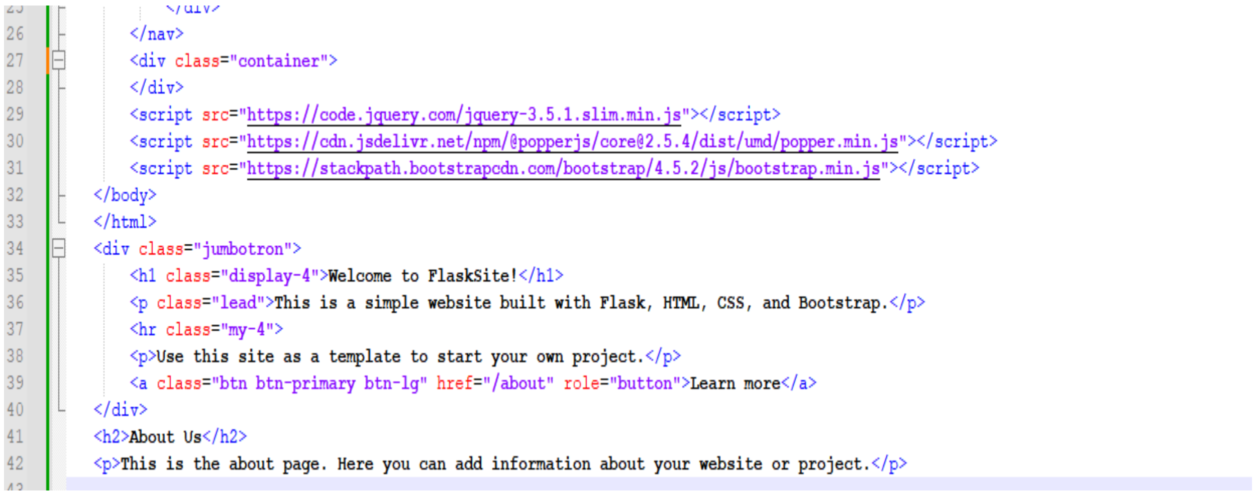
Then we write the Flask application code as shown below:

In [11]:
```python
from flask import Flask, render_template

 app = Flask(__name__)

 @app.route('/')
 def home():
     return render_template('index.html')

 @app.route('/about')
 def about():
     return render_template('about.html')

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

Next: we create a base template to extend for all pages using HTML:

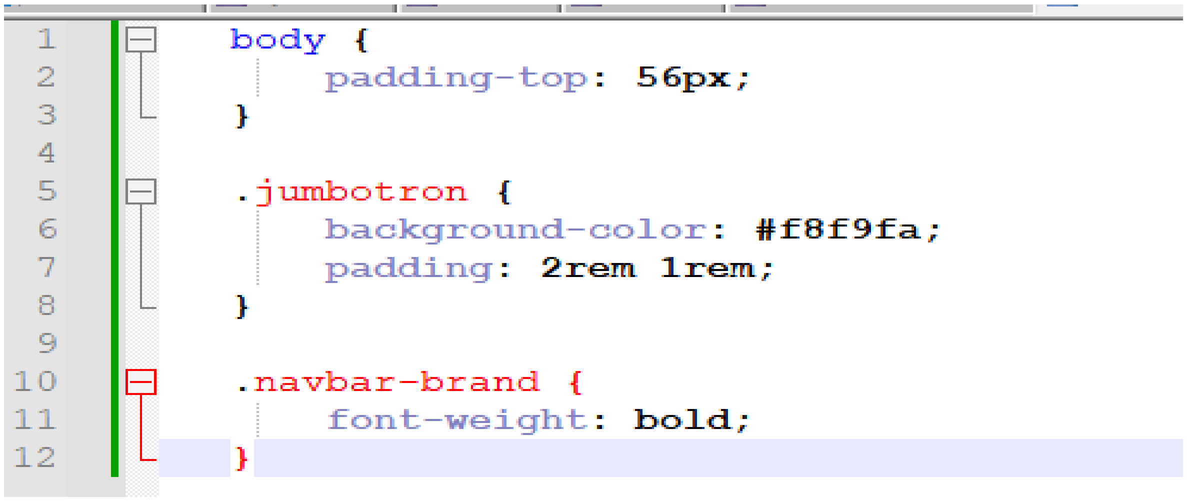


```html
<!DOCTYPE html>
 <html lang="en">
 <head>
     <meta charset="UTF-8">
     <meta name="viewport" content="width=device-width, initial-scale=1.0">
     <title>Simple Flask Website</title>
     <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
     <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
 </head>
 <body>
     <nav class="navbar navbar-expand-lg navbar-light bg-light">
         <a class="navbar-brand" href="#">FlaskSite</a>
         <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNav" aria-controls="navbarNav" aria-expa
             <span class="navbar-toggler-icon"></span>
         </button>
         <div class="collapse navbar-collapse" id="navbarNav">
             <ul class="navbar-nav">
                 <li class="nav-item">
                     <a class="nav-link" href="/">Home</a>
                 </li>
                 <li class="nav-item">
                     <a class="nav-link" href="/about">About</a>
                 </li>
             </ul>
         </div>
     </nav>
     <div class="container">
     </div>
     <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
     <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.5.4/dist/umd/popper.min.js"></script>
     <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
 </body>
```



```
            </div>
        </nav>
        <div class="container">
        </div>
        <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
        <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.5.4/dist/umd/popper.min.js"></script>
        <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
    </body>
    </html>
    <div class="jumbotron">
        <h1 class="display-4">Welcome to FlaskSite!</h1>
        <p class="lead">This is a simple website built with Flask, HTML, CSS, and Bootstrap.</p>
        <hr class="my-4">
        <p>Use this site as a template to start your own project.</p>
        <a class="btn btn-primary btn-lg" href="/about" role="button">Learn more</a>
    </div>
    <h2>About Us</h2>
    <p>This is the about page. Here you can add information about your website or project.</p>
```

Then: we decorate the website using CSS to style it as shown below:

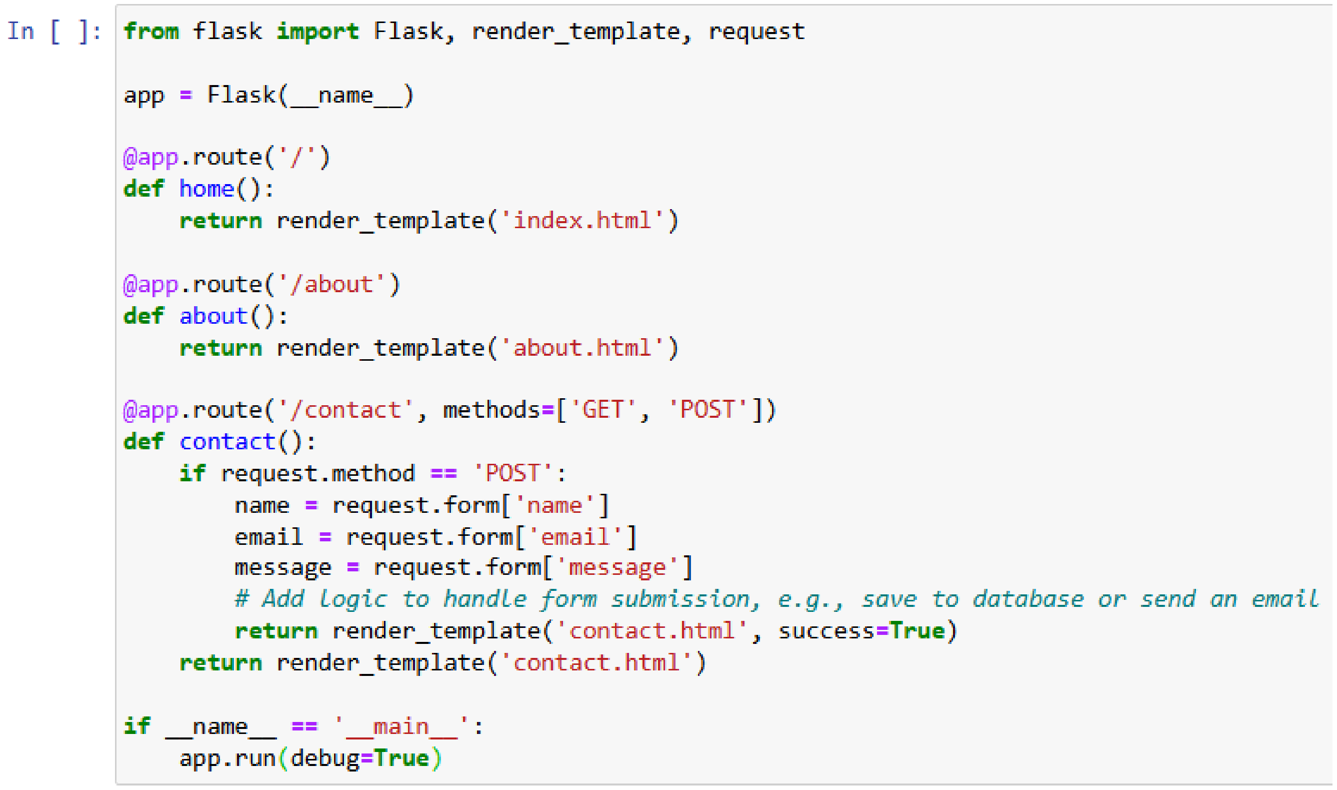


```
body {
    padding-top: 56px;
}

.jumbotron {
    background-color: #f8f9fa;
    padding: 2rem 1rem;
}

.navbar-brand {
    font-weight: bold;
}
```

Next: we run the Run Flask Application , which is :

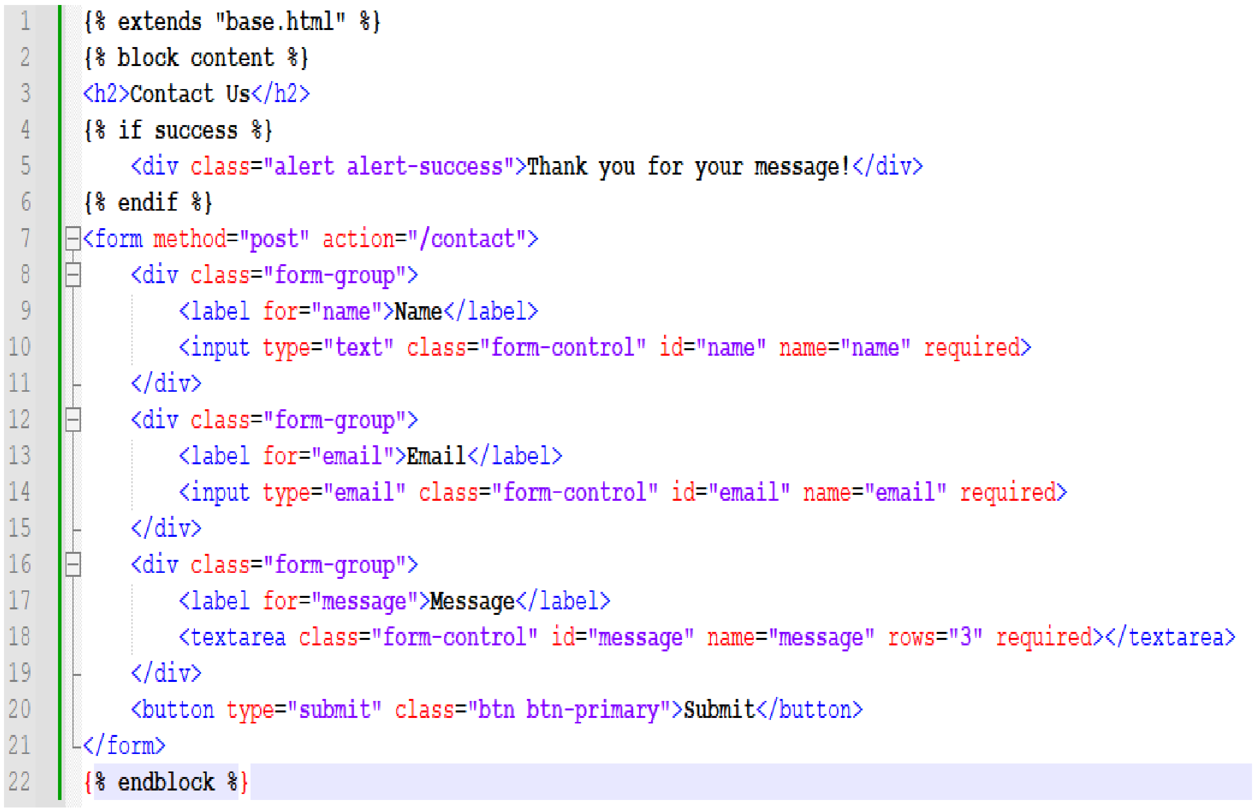
http://127.0.0.1: 5000/.

Then: we Implement Basic server-side Functionality: so to do a contact form we can just add it like this:



In [ ]:
```
from flask import Flask, render_template, request

app = Flask(__name__)

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/about')
def about():
    return render_template('about.html')

@app.route('/contact', methods=['GET', 'POST'])
def contact():
    if request.method == 'POST':
        name = request.form['name']
        email = request.form['email']
        message = request.form['message']
        # Add logic to handle form submission, e.g., save to database or send an email
        return render_template('contact.html', success=True)
    return render_template('contact.html')

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

So if we add the contact app we need to add a contact page with HTML like this:



```
{% extends "base.html" %}
{% block content %}
<h2>Contact Us</h2>
{% if success %}
    <div class="alert alert-success">Thank you for your message!</div>
{% endif %}
<form method="post" action="/contact">
    <div class="form-group">
        <label for="name">Name</label>
        <input type="text" class="form-control" id="name" name="name" required>
    </div>
    <div class="form-group">
        <label for="email">Email</label>
        <input type="email" class="form-control" id="email" name="email" required>
    </div>
    <div class="form-group">
        <label for="message">Message</label>
        <textarea class="form-control" id="message" name="message" rows="3" required></textarea>
    </div>
    <button type="submit" class="btn btn-primary">Submit</button>
</form>
{% endblock %}
```

Finally: using this code make us create a simple, responsive Flask website styled with Bootstrap and implement basic server-side functionality.

End of Homework

Thanks for reading

Soha Yahia Darwish

Maram Mohammed Alabdo