Name: Nour Badour, Number: 2632, Submitted To GitHub:@NourBadour
Name: Soheer Alkadie, Number:2654, Submitted To GitHub:@ SoheerAlkadie

Second Network Programming Homework

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.

```python
import socket
import threading
# Bank account details
accounts = {
    '123456': {'balance': 1000, 'password': 'pass123'},
    '789012': {'balance': 500, 'password': 'pass456'}
}
# Function to handle client requests
def handle_client(client_socket):
    while True:
        # Receive client request
        request = client_socket.recv(1024).decode()
        # Parse request
        req_parts = request.split()
        command = req_parts[0]
        # Authenticate client
        if command == 'LOGIN':
            account_number = req_parts[1]
            password = req_parts[2]
            if account_number in accounts and accounts[account_number]['password
                client_socket.send("Authenticated".encode())
            else:
                client_socket.send("Invalid credentials".encode())
        # Check balance
        elif command == 'BALANCE':
            account_number = req_parts[1]
            if account_number in accounts:
                balance = accounts[account_number]['balance']
                client_socket.send(f"Balance: {balance}".encode())
```

```
        # Check balance
        elif command == 'BALANCE':
            account_number = req_parts[1]
            if account_number in accounts:
                balance = accounts[account_number]['balance']
                client_socket.send(f"Balance: {balance}".encode())
            else:
                client_socket.send("Account not found".encode())
        # Deposit
        elif command == 'DEPOSIT':
            account_number = req_parts[1]
            amount = int(req_parts[2])
            if account_number in accounts:
                accounts[account_number]['balance'] += amount
                client_socket.send("Deposit successful".encode())
            else:
                client_socket.send("Account not found".encode())
        # Withdraw
        elif command == 'WITHDRAW':
            account_number = req_parts[1]
            amount = int(req_parts[2])
            if account_number in accounts:
                if accounts[account_number]['balance'] >= amount:
                    accounts[account_number]['balance'] -= amount
                    client_socket.send("Withdrawal successful".encode())
                else:
                    client_socket.send("Insufficient funds".encode())
            else:
                client_socket.send("Account not found".encode())
```

```
        # Close connection
        elif command == 'EXIT':
            break
    # Send final balance and close connection
    client_socket.send(f"Final balance: {accounts[account_number]['balance']}".e
    client_socket.close()
# Main function to start the server
def main():
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind(('0.0.0.0', 9999))
    server.listen(5)
    print("Server started")
    while True:
        client_socket, addr = server.accept()
        print(f"Connection from {addr}")
        # Start a new thread to handle the client
        client_handler = threading.Thread(target=handle_client, args=(client_soc
        client_handler.start()
if __name__ == "__main__":
    main()
```

```python
import socket
def main():
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect(('localhost', 9999))  # Replace 'localhost' with the server's
    # Login
    account_number = input("Enter your account number: ")
    password = input("Enter your password: ")
    login_request = f"LOGIN {account_number} {password}"
    client.send(login_request.encode())
    response = client.recv(1024).decode()
    print(response)
    # If authenticated, allow banking operations
    if response == "Authenticated":
        while True:
            print("\nChoose operation:")
            print("1. Check balance")
            print("2. Deposit")
            print("3. Withdraw")
            print("4. Exit")
            choice = input("Enter choice: ")
            if choice == '1':
                client.send(f"BALANCE {account_number}".encode())
                balance_response = client.recv(1024).decode()
                print(balance_response)
            elif choice == '2':
                amount = input("Enter amount to deposit: ")
                client.send(f"DEPOSIT {account_number} {amount}".encode())
                deposit_response = client.recv(1024).decode()
                print(deposit_response)
```

```python
            elif choice == '3':
                amount = input("Enter amount to withdraw: ")
                client.send(f"WITHDRAW {account_number} {amount}".encode())
                withdraw_response = client.recv(1024).decode()
                print(withdraw_response)
            elif choice == '4':
                client.send("EXIT".encode())
                final_balance_response = client.recv(1024).decode()
                print(final_balance_response)
                break
            else:
                print("Invalid choice")
    else:
        print("Authentication failed")
    client.close()
if __name__ == "__main__":
    main()
```

```
Enter your account number: 789012
Enter your password: pass456
Authenticated

Choose operation:
1. Check balance
2. Deposit
3. Withdraw
4. Exit
Enter choice: 1
Balance: 500

Choose operation:
1. Check balance
2. Deposit
3. Withdraw
4. Exit
Enter choice: 3
Enter amount to withdraw: 500
Withdrawal successful

Choose operation:
1. Check balance
2. Deposit
3. Withdraw
4. Exit
Enter choice: 4
Final balance: 0
```

```
Enter your account number: 123456
Enter your password: pass123
Authenticated

Choose operation:
1. Check balance
2. Deposit
3. Withdraw
4. Exit
Enter choice: 1
Balance: 1000

Choose operation:
1. Check balance
2. Deposit
3. Withdraw
4. Exit
Enter choice: 2
Enter amount to deposit: 500
Deposit successful

Choose operation:
1. Check balance
2. Deposit
3. Withdraw
4. Exit
Enter choice: 4
Final balance: 1500
```

```
Server started
Connection from ('127.0.0.1', 54281)
Connection from ('127.0.0.1', 54357)
Connection from ('127.0.0.1', 54373)
```

السؤال الأول:
السيرفر:

تم انشاء خادمًا بنكيًا باستخدام مكتبة socket و threading الخيوط في بايثون. يتم تخزين تفاصيل الحسابات البنكية في معجم يحتوي على أرقام الحسابات، الأرصدة، وكلمات المرور.

تتولى دالة "handle_client" معالجة طلبات العملاء المتصلة بالخادم. تبدأ الدالة بتلقي طلبات العميل، تحليلها، وتحديد نوع الأمر المطلوب تنفيذه.

أوامر العميل تشمل:

– تسجيل الدخول: يقوم العميل بإرسال أمر "LOGIN" متبوعًا برقم الحساب وكلمة المرور. إذا كانت البيانات صحيحة، يتم إعلام العميل بنجاح المصادقة، وإلا يتم إخباره بأن البيانات غير صحيحة.

– التحقق من الرصيد: يرسل العميل أمر "BALANCE" متبوعًا برقم الحساب، فيتم إرسال الرصيد الحالي للعميل.

– الإيداع: يرسل العميل أمر "DEPOSIT" متبوعًا برقم الحساب والمبلغ، فيتم زيادة الرصيد بالمبلغ المودع وإعلام العميل بنجاح العملية.

– السحب: يرسل العميل أمر "WITHDRAW" متبوعًا برقم الحساب والمبلغ، ويتم التحقق من توفر الرصيد الكافي، ثم خصم المبلغ من الرصيد إذا كان كافيًا، أو إعلام العميل بنقص الرصيد.

– إنهاء الاتصال: يرسل العميل أمر "EXIT" لإنهاء الجلسة، حيث يتم إرسال الرصيد النهائي للعميل قبل إغلاق الاتصال.

الدالة الرئيسية "main" تقوم بإعداد الخادم، إنشاء مقبس الشبكة، وربطه بعنوان IP والمنفذ المحددين. يبدأ الخادم بالاستماع لطلبات الاتصال، وعند تلقي اتصال من عميل جديد، يتم قبول الاتصال وإنشاء خيط جديد للتعامل مع هذا العميل، مما يسمح بالخادم بمعالجة عدة عملاء في نفس الوقت.

العميل:

البرنامج ينشئ عميلًا بنكيًا يتصل بخادم عبر مكتبة socket في بايثون. يبدأ العميل بإنشاء مقبس شبكة والاتصال بالخادم على العنوان المحلي والمنفذ 9999. يطلب من المستخدم إدخال رقم الحساب وكلمة المرور، ثم يرسل هذه المعلومات إلى الخادم لتسجيل الدخول.

إذا كانت المصادقة ناجحة، يمكن للمستخدم تنفيذ عمليات بنكية تشمل:

- التحقق من الرصيد عن طريق إرسال أمر "BALANCE" متبوعًا برقم الحساب واستقبال الرد من الخادم.
- الإيداع عن طريق إرسال أمر "DEPOSIT" متبوعًا برقم الحساب والمبلغ واستقبال تأكيد العملية.
- السحب عن طريق إرسال أمر "WITHDRAW" متبوعًا برقم الحساب والمبلغ، واستقبال تأكيد العملية أو إشعار بنقص الرصيد.
- إنهاء الجلسة عن طريق إرسال أمر "EXIT"، واستقبال الرصيد النهائي قبل إغلاق الاتصال.

إذا فشلت المصادقة، يتم إبلاغ المستخدم بفشل العملية، ويتم إنهاء الاتصال بالخادم وإغلاق مقبس الشبكة.

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. 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 .

index.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Learn Audio and Video Basics in Communications</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/bootstrap.min.css') }}">
    <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
</head>
<body>
    <header>
        <nav>
            <ul>
                <li><a href="{{ url_for('index') }}">Home</a></li>
                <li><a href="{{ url_for('about') }}">About</a></li>
                <li><a href="{{ url_for('contact') }}">Contact</a></li>
            </ul>
        </nav>
    </header>

    <div class="content">
        <h1>Learn Audio and Video Basics</h1>
        <p>Start your journey to understanding the fundamentals of audio and video in communications.</p>
        <p>Here are some key features of our website:</p>
        <ul>
            <li>Lessons on audio and video encoding, data compression, and audiovisual techniques.</li>
            <li>Practical applications for designing and developing audio and video systems in communications.</li>
            <li>Educational resources on audio and video applications in online and mobile communications.</li>
            <li>Forum for discussing challenges and innovations in audio and video communications.</li>
        </ul>
        <p>Join us today and explore the world of audio and video in communications!</p>
    </div>
</body>
</html>
```

index.html | about.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Learn Audio and Video Basics in Communications - About</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/bootstrap.min.css') }}">
    <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
</head>
<body>
    <header>
        <nav>
            <ul>
                <li><a href="{{ url_for('index') }}">Home</a></li>
                <li><a href="{{ url_for('about') }}">About</a></li>
                <li><a href="{{ url_for('contact') }}">Contact</a></li>
            </ul>
        </nav>
    </header>

    <div class="content">
        <h1>About Learn Audio and Video Basics</h1>
        <p>This platform provides comprehensive knowledge and practical skills in audio and video in communications.</p>
        <p>Our goal is to equip learners with understanding of audio and video encoding, data compression, and applications.</p>
        <p>We offer detailed lessons, practical projects, resources, and a community forum.</p>
        <p>Join us today to explore the fundamentals of audio and video in communications!</p>
    </div>

</body>
</html>
```

index.html | about.html | contact.html

```html
<!DOCTYPE html>
<html>
<head>
    <title>Learn Audio and Video Basics in Communications - Contact</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/bootstrap.min.css') }}">
    <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
</head>
<body>
    <header>
        <nav>
            <ul>
                <li><a href="{{ url_for('index') }}">Home</a></li>
                <li><a href="{{ url_for('about') }}">About</a></li>
                <li><a href="{{ url_for('contact') }}">Contact</a></li>
            </ul>
        </nav>
    </header>

    <div class="content">
        <h1>Contact Us</h1>
        <p>Have a question or need assistance? Fill out the form below to get in touch with our team.</p>

        <form action="/submit_contact" method="POST">
            <label for="name">Name:</label>
            <input type="text" id="name" name="name" required>

            <label for="email">Email:</label>
            <input type="email" id="email" name="email" required>

            <label for="message">Message:</label>
            <textarea id="message" name="message" required></textarea>

            <input type="submit" value="Submit">
        </form>
    </div>
</body>
</html>
```

```python
from flask import Flask, render_template

app = Flask(__name__)

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

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

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



if __name__ == '__main__':
    app.run(debug=True , port=3245)
```

يتم إنشاء تطبيق Flask باستخدام الكود app = Flask(__name__).

يتم تمرير __name__ كمعامل لتحديد اسم التطبيق وتحديد موقع ملفات html.

يتم تعريف المسارات (routes) باستخدام المزخرف @app.route

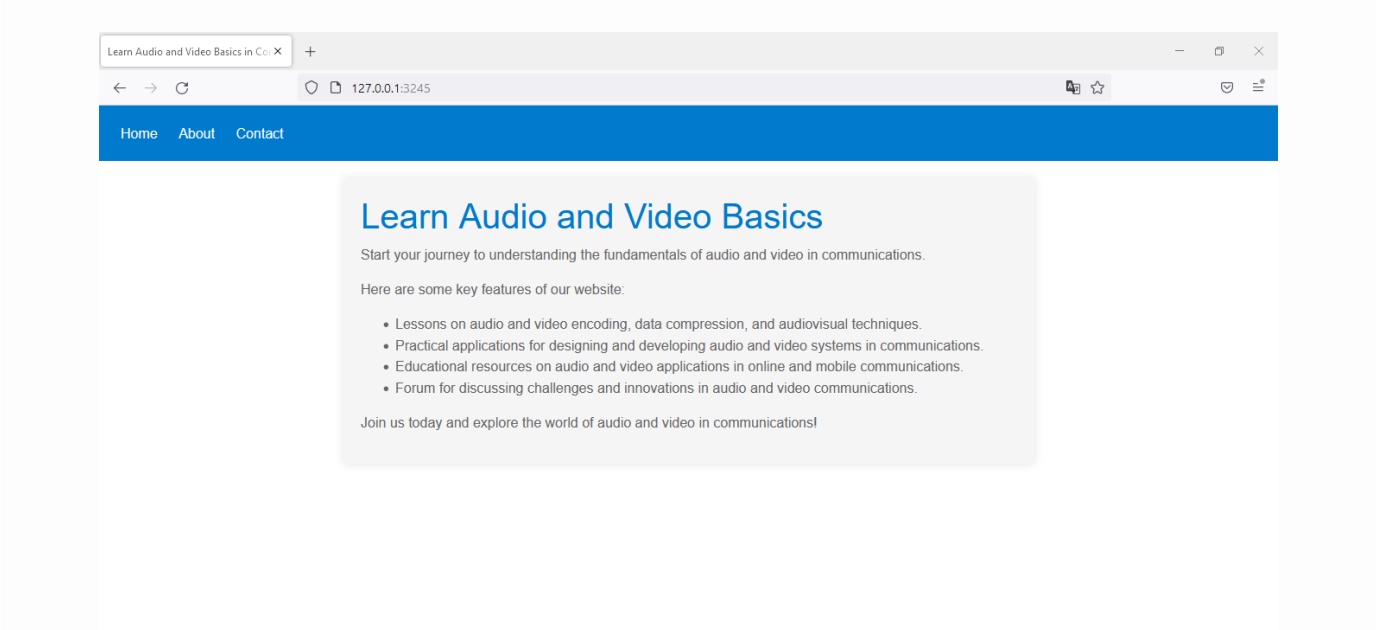
المسار /template يعود إلى الصفحة الرئيسية ويتم تعيينه لدالة home().

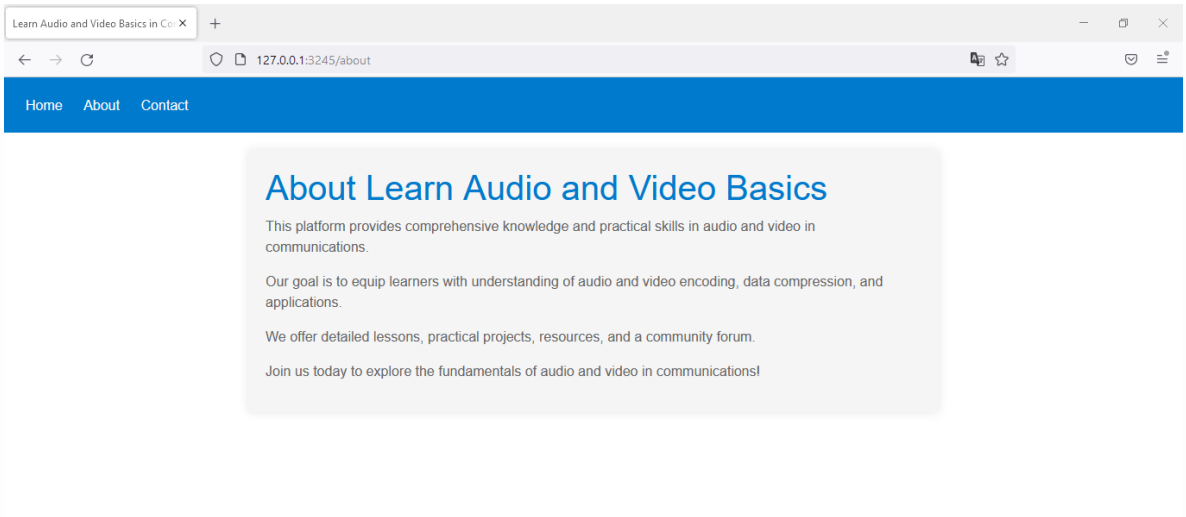
المسار /template/about يعود إلى صفحة "about" ويتم تعيينه لدالة about().

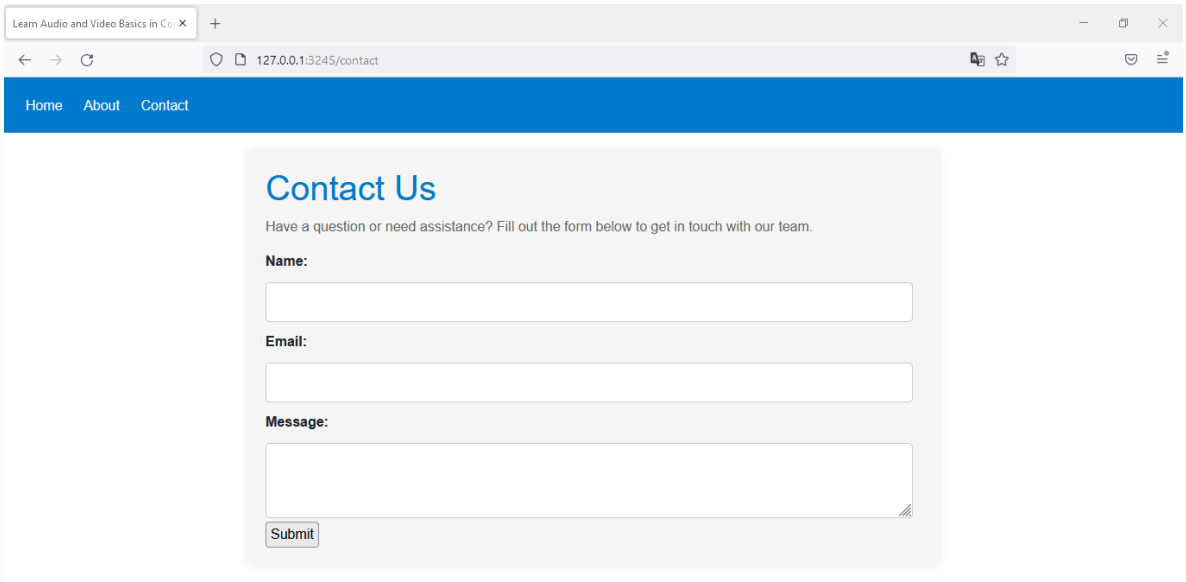
المسار /template/contact يعود إلى صفحة " contact" ويتم تعيينه لدالة contact().

إذا كان البرنامج يتم تشغيله مباشرة عن طريق تشغيل البرنامج الرئيسي ، فإنه يشغل التطبيق بتفعيل وضع التصحيح (debug mode) بواسطة الأمر app.run(debug=True).

يتم التشغيل على port 3245




Learn Audio and Video Basics in Co
127.0.0.1:3245/about
Home
About
Contact
About Learn Audio and Video Basics
This platform provides comprehensive knowledge and practical skills in audio and video in communications.
Our goal is to equip learners with understanding of audio and video encoding, data compression, and applications.
We offer detailed lessons, practical projects, resources, and a community forum.
Join us today to explore the fundamentals of audio and video in communications!


Learn Audio and Video Basics in Co
127.0.0.1:3245/contact
Home
About
Contact
Contact Us
Have a question or need assistance? Fill out the form below to get in touch with our team.
Name:
Email:
Message:
Submit