Syrian Arab Republic

Lattakia - Tishreen University

Department of Communication and electrical engineering

5th , Network Programming :

Homework No2

الجمهورية العربية السورية

اللاذقية – جامعـة تشريــن

كلية الهندسة الكهربائية والميكانيكية

قسم هندسة الاتصالات والالكترونيات

السنة الخامسة: وظيفة2 برمجة شبكات

Second Network Programming Homework

الوظيفة الثانية برمجة شبكات

تقدمة الطلاب

Names:

- Nawar Atfeh **نوار تمام عطفه 2593 ,**
- Bashar Ismail **بشار حسين اسماعيل 3001 ,**

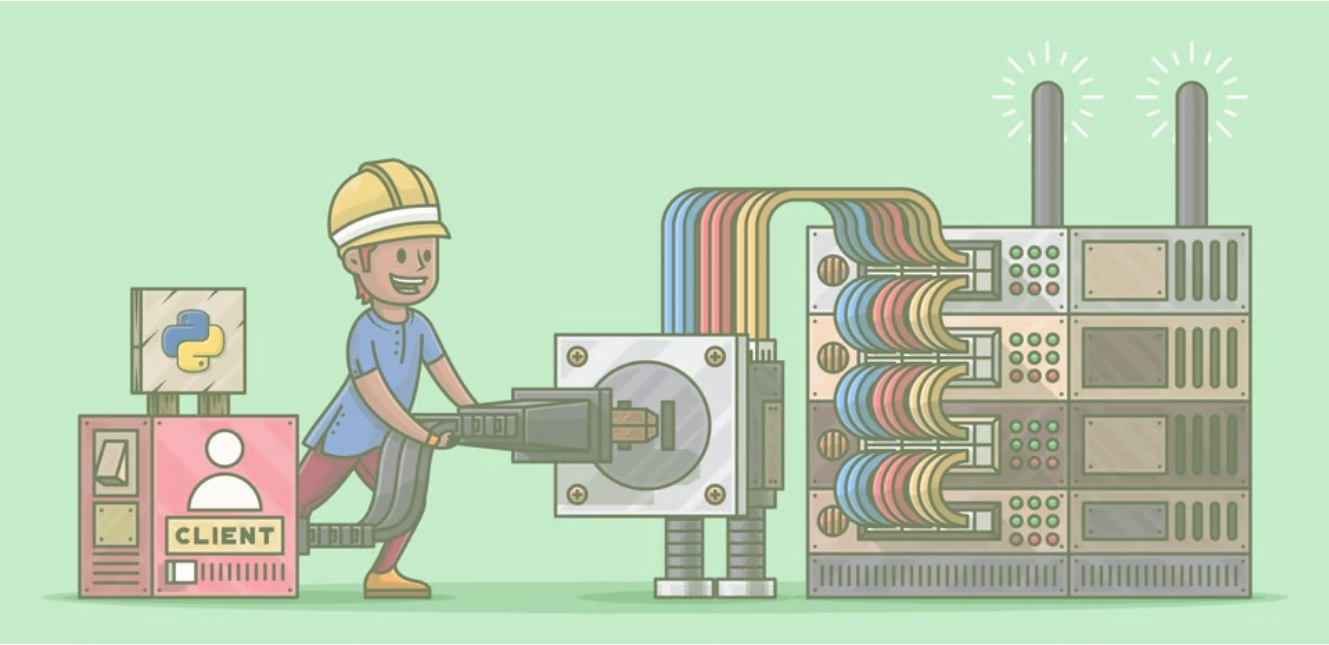
Question 1: Bank ATM Application with TCP Server/Client and Multi-threading

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.



وصف المشكلة:

نحن بحاجة إلى إنشاء تطبيق Bank ATM حيث يمكن للعملاء (المستخدمين) الاتصال بخادم TCP server. يجب أن يتعامل الخادم مع العديد من العملاء في نفس الوقت اعتماداً على (multi-threading). سيتصل كل عميل بالخادم ويقوم بعمليات مصرفية مثل التحقق من الرصيد (check balance) وإيداع الأموال (deposit) وسحب الأموال(withdraw). سيحتفظ الخادم بتفاصيل الحساب وتحديث الأرصدة وفقًا لذلك.

بناء خوارزمية للحل:

1- تهيئة السيرفر. Server Initialization

2 - التعامل مع اتصال العميل Client Connection Handling

3- استخدام Multi-threading

4- ادارة الحساب Account Management

Solution:

1- Set up the basic server and client using Python's socket module:

» Basic server:

```
import socket

#Define start_server() function :responsible for setting up and running the TCP server.
def start_server():
    # Create a TCP socket object:
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # Allows the socket address to be reused immediately after the socket is closed:
    server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR,1)
    # bind the server socket to a specific address and port.
    server_socket.bind(('0.0.0.0', 1234)) #(0.0.0.0 means  all available network interfaces
    # Start listening for incoming connections:
    server_socket.listen()
    print("Server is listening on port 1234...")
    connected = True

    # Accepting Client Connections
    while connected:
        client_socket, client_address = server_socket.accept()
        print(f"Connection from {client_address} has been established!")
        msg_from_client = client_socket.recv(1024).decode("utf-8")
        client_socket.send(f"Hello {msg_from_client}!, Welcome to the bank ATM!".encode("utf-8"))
        client_socket.close()


if __name__ == "__main__":
    start_server()
```

» Basic client:

```python
import socket
# Define connect_to_server() function :  responsible for establishing a connection to the server.
def connect_to_server():
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # If the client socket doesn't receive any data within 10 seconds, raise a socket timeout exception:
    client_socket.settimeout(10)
    msg_from_client = input("Enter a name to send to the server: ")
    try:
        client_socket.connect(('127.0.0.1', 1234))
        client_socket.send(msg_from_client.encode("utf-8"))
        msg_from_server = client_socket.recv(1024) # receive data from the server
        print(msg_from_server.decode("utf-8"))
    # Handling Connection Errors:
    except socket.error as err:
        print('exp', err)
    #Ensures that the client socket is always closed
    finally:
        client_socket.close()


if __name__ == "__main__":
    connect_to_server()
```

Connection:

server_side:

```
Server is listening on port 1234...
Connection from ('127.0.0.1', 60684) has been established!
Connection from ('127.0.0.1', 60686) has been established!
Connection from ('127.0.0.1', 60689) has been established!
```

client_1:

```
Enter a name to send to the server: c1
Hello c1!, Welcome to the bank ATM!
```

client_2:

```
Enter a name to send to the server: c2
Hello c2!, Welcome to the bank ATM!
```

client_3:

```
Enter a name to send to the server: c3
Hello c3!, Welcome to the bank ATM!
```

2- Improve the basic server/ client to handle multiple client connection using multi-threading, and manage banking operations:

» server.py:

```python
# import the necessary modules for creating a socket server and using threading for concurrent connections:
import socket
import threading

# Create a dictionary 'accounts' stores the pre-defined account information. Each account is represented by a
username, and the corresponding value is a dictionary containing the balance and password.
accounts = {
    'client1': {'balance': 1000, 'password': 'password1'},
    'client2': {'balance': 2000, 'password': 'password2'},
    'client3': {'balance': 1500, 'password': 'password3'},
}
# Define Two functions 'check_balance' to return the account balance, and 'deposit' to perform a deposit.
def check_balance(accounts, account_username):
    balance = accounts[account_username]['balance']
    return balance
# deposit function take the 'accounts' dictionary, account username, and amount as parameters and return the updated
balance.
def deposit(accounts, account_username, amount):
    balance = accounts[account_username]['balance']
    balance += amount
    return balance
```

◈ شرح لما سبق:

1- استدعاء المكاتب الضرورية لإنشاء TCP Socket والتعامل مع عدة اتصالات باستخدام multi-threading

2- تعريف قاموس dictionary، والذي يمثل قاعدة بيانات لاحتواء الحسابات الموجودة وكلمة السر والرصيد الخاصين بكل حساب.

3- تعريف تابع check_balance الذي يقوم بعملية تفقد الرصيد وعرض المبلغ الموجود في الرصيد عند استدعاءه، وتعريف التابع deposit والذي يقوم بعملية الإيداع، يقوم بزيادة الرصيد بالمبلغ الذي اودعه المستخدم عند استدعاءه.

```python
# Function to handle client connections
def handle_client(client_socket, client_address):
    print(f"Connection from {client_address} has been established!")
    client_socket.send(f" Welcome to the bank ATM!".encode("utf-8"))

    # Authentication process
    client_socket.send("Enter your account username: ".encode("utf-8"))
    account_username = client_socket.recv(1024).decode("utf-8")

    client_socket.send("Enter your password: ".encode("utf-8"))
    password = client_socket.recv(1024).decode("utf-8")

    # Check if account exists and password matches
    if account_username.strip().lower() in accounts and accounts[account_username]['password'] == password:
        client_socket.send("Authentication successful!".encode("utf-8"))
        authenticated = True
    else:
        client_socket.send("Authentication failed! Closing connection....".encode("utf-8"))
        client_socket.close()
        return # the function should terminate immediately without executing any further code

    while authenticated:
        client_socket.send("Choose operation: \n1. Check Balance \n2. Deposit \n3. Withdraw \n4. Exit".encode("utf-8"))
        operation = client_socket.recv(1024).decode("utf-8")

        # Check balance:
        if operation == '1':
            balance = check_balance(accounts, account_username)
            client_socket.send(f"Your balance is: ${balance}".encode("utf-8"))
        # Deposit:
        elif operation == '2':
            client_socket.send("Enter amount to deposit: ".encode("utf-8"))
            amount = float(client_socket.recv(1024).decode("utf-8"))
            balance = deposit(accounts,account_username, amount)
            accounts[account_username]['balance'] = balance
            client_socket.send(f"${amount} deposited. New balance is: ${balance}".encode( "utf-8"))
        # Withdraw:
        elif operation == '3':
            client_socket.send("Enter amount to withdraw: ".encode("utf-8"))
            amount = float(client_socket.recv(1024).decode("utf-8"))
            balance = accounts[account_username]['balance']
            if amount <= accounts[account_username]['balance']:
                accounts[account_username]['balance'] -= amount
                client_socket.send(bytes(f"${amount} withdrawn. New balance is: ${accounts[account_username]['balance']}", "utf-8"))
            else:
                client_socket.send(bytes("Insufficient funds!", "utf-8"))
        elif operation == '4':
            client_socket.send(f"Final balance: ${accounts[account_username]['balance']}. Goodbye!".encode("utf-8"))
            client_socket.close()
            break
        else:
            client_socket.send("Invalid operation. Please try again.".encode("utf-8"))
```

◈ شرح لما سبق:

1- تعريف تابع handle_client وظيفته إدارة عمليات العملاء داخل تطبيق البنك كالتالي:

⁂ تأكيد تمام عملية الاتصال مع العميل، وارسال رسالة ترحيبية وطلب اسم المستخدم وكلمة المرور من المستخدم.

⁂ التحقق من أن صحة البيانات المدخلة، وفي حال تمت المصادقة، يرسل السيرفر للعميل قائمة بالعمليات الحسابية ليختار العملية التي يريد القيام بها(الاطلاع على قيمة الحساب الحالية "1" أوالإيداع "2" أو السحب "3" مع الأخذ بعين الاعتبار إظهار رسالة خطأ عند سحب قيمة أكبر من الموجودة، أو الخروج "4"، أو ارسال رسالة خطأ في حال اختيار المستخدم عملية غير موجودة.

⁂ ارسال رسالة بالرصيد المتبقي، ورسالة وداع في حال اختيار العميل الخروج، ومن ثم اغلاق الاتصال بين الخادم والعميل في نهاية التابع.

```python
#Define start_server() function :responsible for setting up and running the TCP server.
def start_server():
    # Create a TCP socket object:
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # Allows the socket address to be reused immediately after the socket is closed:
    server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR,1)
    # bind the server socket to a specific address and port.
    server_socket.bind(('0.0.0.0', 1234)) #(0.0.0.0 means  all available network interfaces
    # Start listening for incoming connections:
    server_socket.listen()
    print("Server is listening on port 1234...")
    connected = True

    # Accepting Client Connections
    while connected:
        client_socket, client_address = server_socket.accept()
        # Create a new thread to handle the client connection
        client_thread = threading.Thread(target=handle_client, args=(client_socket, client_address))
        client_thread.start()


if __name__ == "__main__":
    start_server()
```

◈ شرح لما سبق:

1- تعريف التابع start_server الذي يقوم بما يلي:

⁂ انشاء السوكيت ببروتوكول TCP/IPV4 لإنشاء اتصال آمن.

⁂ ربط الخادم بعنوان IP ومنفذ 1234.

⁂ الاستماع للطلبات القادمة من العملاء، وارسال رسالة للاعم بأن السيرفر يتنصت على المنفذ 1234.

⁂ قبول الاتصالات بشكل متواصل، وانشاء سوكيت وعنوان للعميل باستخدام التابع accept.

⁂ انشاء client_thread جديدة خاصة بكل عميل للسماح للبرنامج بمعالجة طلبات عدة عملاء بوقت واحد.

الكود السابق يجهز الخادم للتعامل مع عدة طلبات بشكل متزامن، دون الحاجة لمعالجة كل طلب على حدى بشكل متتابع.

» client.py:

```
import socket
# Define connect_to_server() function :  responsible for establishing a connection to the server.
def connect_to_server():
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        client_socket.connect(('127.0.0.1', 1234))
        welcome_message = client_socket.recv(1024)
        print(welcome_message.decode("utf-8"))

        # Authentication process:
        # Enter account number and password
        account_username = input(client_socket.recv(1024).decode("utf-8"))
        client_socket.send(account_username.encode("utf-8"))
        password = input(client_socket.recv(1024).decode("utf-8"))
        client_socket.send(password.encode("utf-8"))
        auth_response = client_socket.recv(1024).decode("utf-8")
        print(auth_response)
        if "Authentication successful" in auth_response:
            while True:
                options = client_socket.recv(1024).decode("utf-8")
                operation = input(f"{options}: ")
                client_socket.send(operation.encode("utf-8"))
                if operation == '1':
                    response = client_socket.recv(1024).decode("utf-8")
                    print(f"{response}\n")
                elif operation == '2':
                    amount = input(client_socket.recv(1024).decode("utf-8"))
                    client_socket.send(amount.encode("utf-8"))
                    response = client_socket.recv(1024).decode("utf-8")
                    print(f"{response}\n")
                elif operation == '3':
                    amount = input(client_socket.recv(1024).decode("utf-8"))
                    client_socket.send(amount.encode("utf-8") )
                    response = client_socket.recv(1024).decode("utf-8")
                    print(f"{response}\n")
                elif operation == '4':
                    response = client_socket.recv(1024).decode("utf-8")
                    print(response)
                    client_socket.close()
                    break
                else:
                    response = client_socket.recv(1024).decode("utf-8")
                    print(response)
    # Handling Connection Errors:
    except socket.error as err:
        print('exp', err)
    #Ensures that the client socket is always closed
    finally:
        client_socket.close()


if __name__ == "__main__":
    connect_to_server()
```

◈ **شرح كود العميل** client.py:

1- تعريف التابع connenct_to_server المسؤول عن انشاء الاتصال مع السيرفر:

⁂ انشاء TCP socket والاتصال مع السيرفر على ال Local host وعلى المنفذ 1234.

⁂ استقبال وطباعة رسالة الترحيب الوادرة من السيرفر.

⁂ ارسال اسم المستخدم وكلمة المرور الى السيرفر من أجل التحقق من المصادقة .

⁂ في حال تمام المصادقة، يتم الدخول في حلقة من أجل القيام بالعمليات البنكية (تفقد الرصيد، إيداع، سحب، أو خروج)، وارسال العملية المختارة الى السيرفر.

⁂ استقبال نتيجة العملية التي تم اختيارها.

⁂ اغلاق السوكيت وتحرير الموارد.

الكود السابق يمثل عميل بسيط يمكنه الاتصال بخادم أجهزة الصراف الآلي الخاصة بالبنك والمصادقة وتنفيذ العمليات المصرفية الأساسية من خلال سلسلة من الطلبات والاستجابات.

› From server.py:

```
Server is listening on port 1234...
Connection from ('127.0.0.1', 51216) has been established!
Connection from ('127.0.0.1', 51218) has been established!
```

› From client1.py:

```
Enter your account username: client1
Enter your password: password1
Authentication successful!
Choose operation:
1. Check Balance
2. Deposit
3. Withdraw
4. Exit: 1
Your balance is: $1000

Choose operation:
1. Check Balance
2. Deposit
3. Withdraw
4. Exit: 3
Enter amount to withdraw: 500
$500.0 withdrawn. New balance is: $500.0

Choose operation:
1. Check Balance
2. Deposit
3. Withdraw
4. Exit: 4
Final balance: $500.0. Goodbye!
```

› From client2.py:

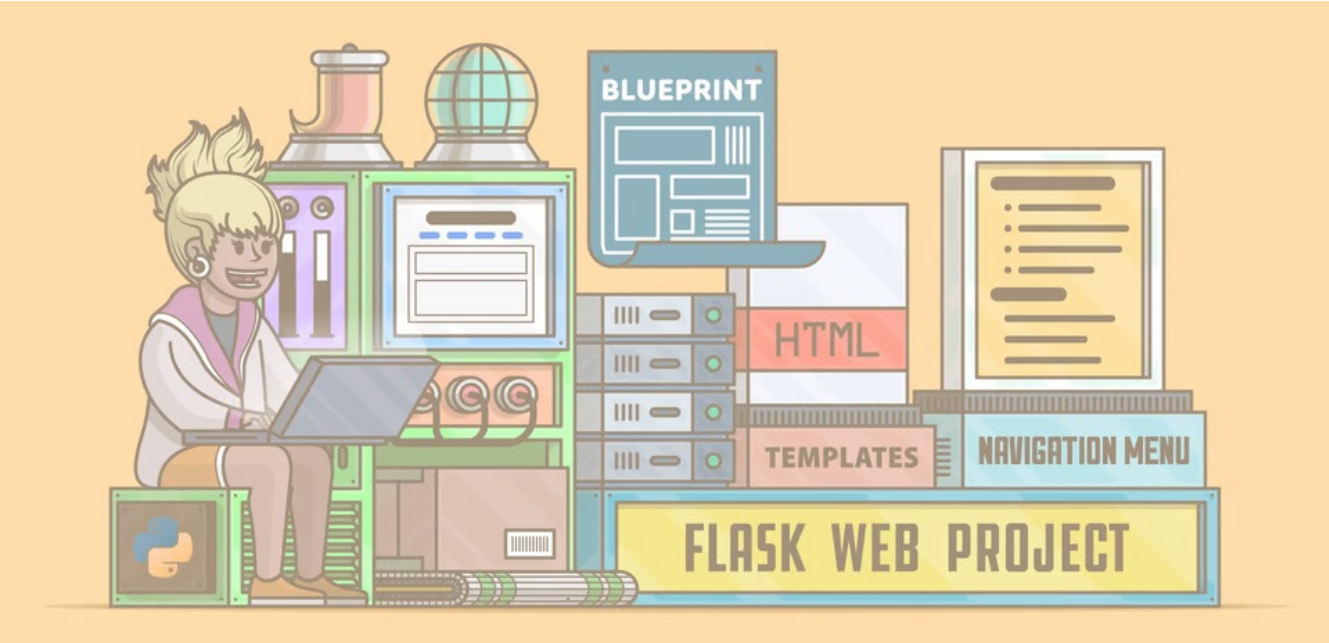
```
 Welcome to the bank ATM!
Enter your account username: client2
Enter your password: password2
Authentication successful!
Choose operation:
1. Check Balance
2. Deposit
3. Withdraw
4. Exit: 1
Your balance is: $2000

Choose operation:
1. Check Balance
2. Deposit
3. Withdraw
4. Exit: 2
Enter amount to deposit: 2000
$2000.0 deposited. New balance is: $4000.0

Choose operation:
1. Check Balance
2. Deposit
3. Withdraw
4. Exit: 4
Final balance: $4000.0. Goodbye!
```

Question 2: Simple Website Project with Python Flask Framework:

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



» flask_website.py:

```python
#Flask is a micro web framework written in Python. It is designed to be simple and easy to use, allowing developers to quickly build web applications.
# import 'Flask' from the flask module to create a Flask application instance.
# import 'render_template' to render HTML templates.
from flask import Flask, render_template

# Create an instance of the Flask class and assigned to the variable 'app':
app = Flask(__name__) #The argument __name__ is passed to the Flask class to let Flask know where to look for templates and static files.


@app.route('/') # Defines the route for the home page ('/')
def home(): # called when a user visits the home page.
    return render_template('home.html') # renders and returns the 'home.html' template.

@app.route('/about') # Defines the route for the about page (/about).
def about(): # Called when a user visits the about page.
    return render_template('about.html') # renders and returns the 'about.html' template.

@app.route('/contact_us') # Defines the route for the contact us page (/contact_us).
def contact_us(): # called when a user visits the contact us page.
    return render_template('contact.html') # renders and returns the 'contact.html' template.
@app.route('/n_git')
def nawar_github():
    return render_template('nawar_git.html')
@app.route('/b_git')
def bashar_github():
    return render_template('bashar_git.html')
@app.route('/ref')
def references():
    return render_template('references.html')
@app.route('/learn_more')
def learnMore():
    return render_template('learn_more.html')
# Ensures that the Flask application runs only if the script is executed directly (not imported as a module):
if __name__ == '__main__':
    app.run(debug=True) # starts the Flask development server with debugging enabled. Debug mode allows the server to reload automatically when code changes, and provides detailed error messages.
```

◈ **الشرح :**

1- قبل الدخول الى الكود، قمنا بإنشاء مجلدين templates لنضع فيه ملفات ال html التي نريد عرضها (يقوم التابع render_template بالبحث افتراضياً ضمن هذا المجلد عن صفحات الhtml، والمجلد الآخر هو static والذي سنضع فيه الصور وملفات ال css، وقمنا بتنزيل مكتبة flask التي تسمح بانشاء تطبيقات الويب باستخدام بايثون.

2- استدعاء Flask والذي يمثل تطبيق Flask، ونستطيع اشتقاق أغراض منه وتعديلها وتشغيلها، واستدعاء render_template الذي هو تابع ضمن مكتبة flask والذي يستخدم لعرض صفحات html.

3- انشاء غرض من الكلاس Flask وتخزينه في المتغير app.

4- تحديد المسار المستخدم للوصول الى الصفحة الرئيسية home ("/")، يتم استدعاء التابع home عندما يقوم المستخدم بزيارة، أو الضغط على الصفحة الرئيسة Home.

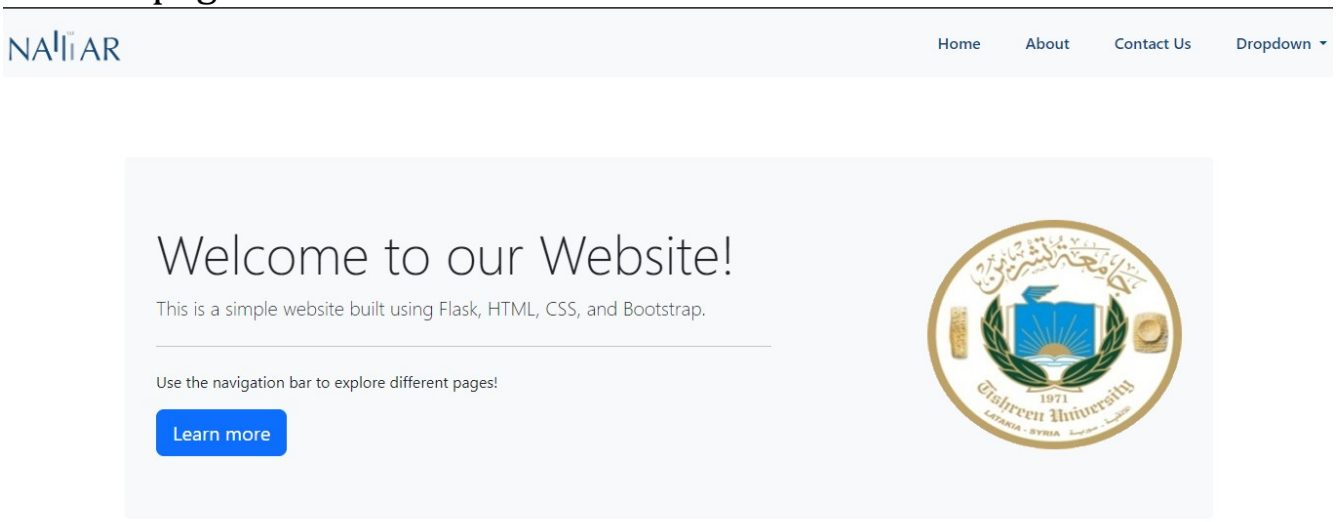
5- عند زيارة الصفحة الرئيسة، يتم استدعاء التابع home ،والذي بدوره يستدعي التابع render_template('home.html') الذي يقوم بارجاع صفحة ال home.html الموجودة في مجلد ال templates.

6- عند زيارة صفحة حول about، يتم استدعاء التابع about ،والذي بدوره يستدعي التابع render_template('about.html') الذي يقوم بارجاع صفحة ال about.html الموجودة في مجلد ال templates، والتي تحوي معلومات حول أسمائنا وأرقامنا الجامعية.

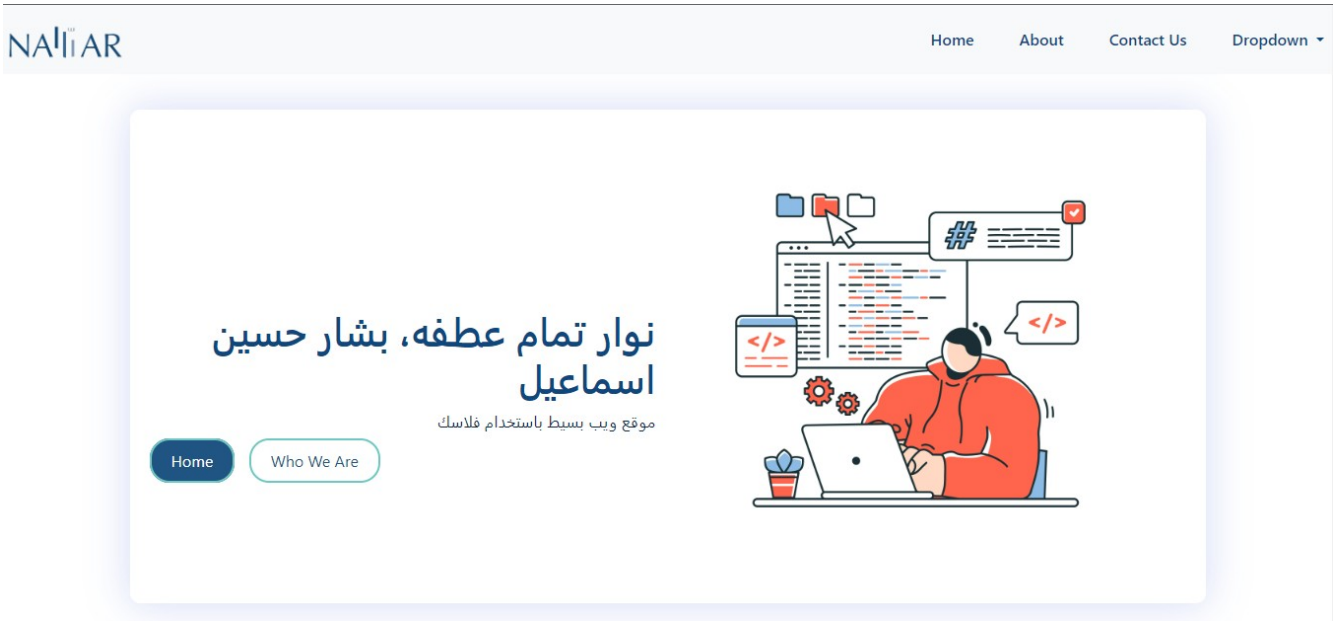
7- عند زيارة صفحة حول Contact Us، يتم استدعاء التابع contact_us ،والذي بدوره يستدعي التابع render_template('contact.html') الذي يقوم بارجاع صفحة ال contact.html الموجودة في مجلد ال templates، والتي تحوي معلومات الاتصال الخاصة بنا.

8- عند الضغط على كلمة Dropdown ستظهر قائمة منسدلة تحوي رابط حساب نوار وبشار على ال github، وتحوي على learn more التي تعرض صفحة باللغة العربية عن المشروع، وتحوي على المراجع references التي تم الاعتماد عليها لانشاء هذا المشروع.

› Home page:

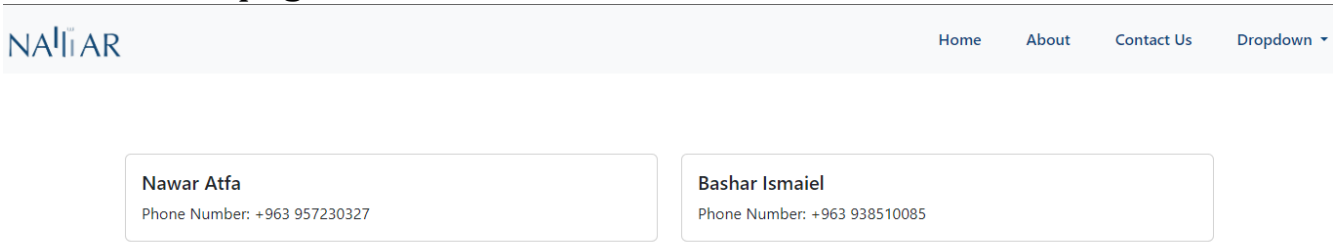


› Learn more page:

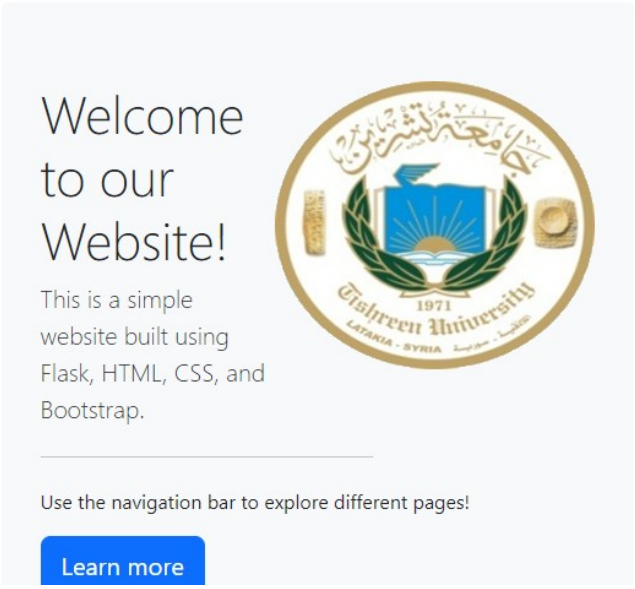


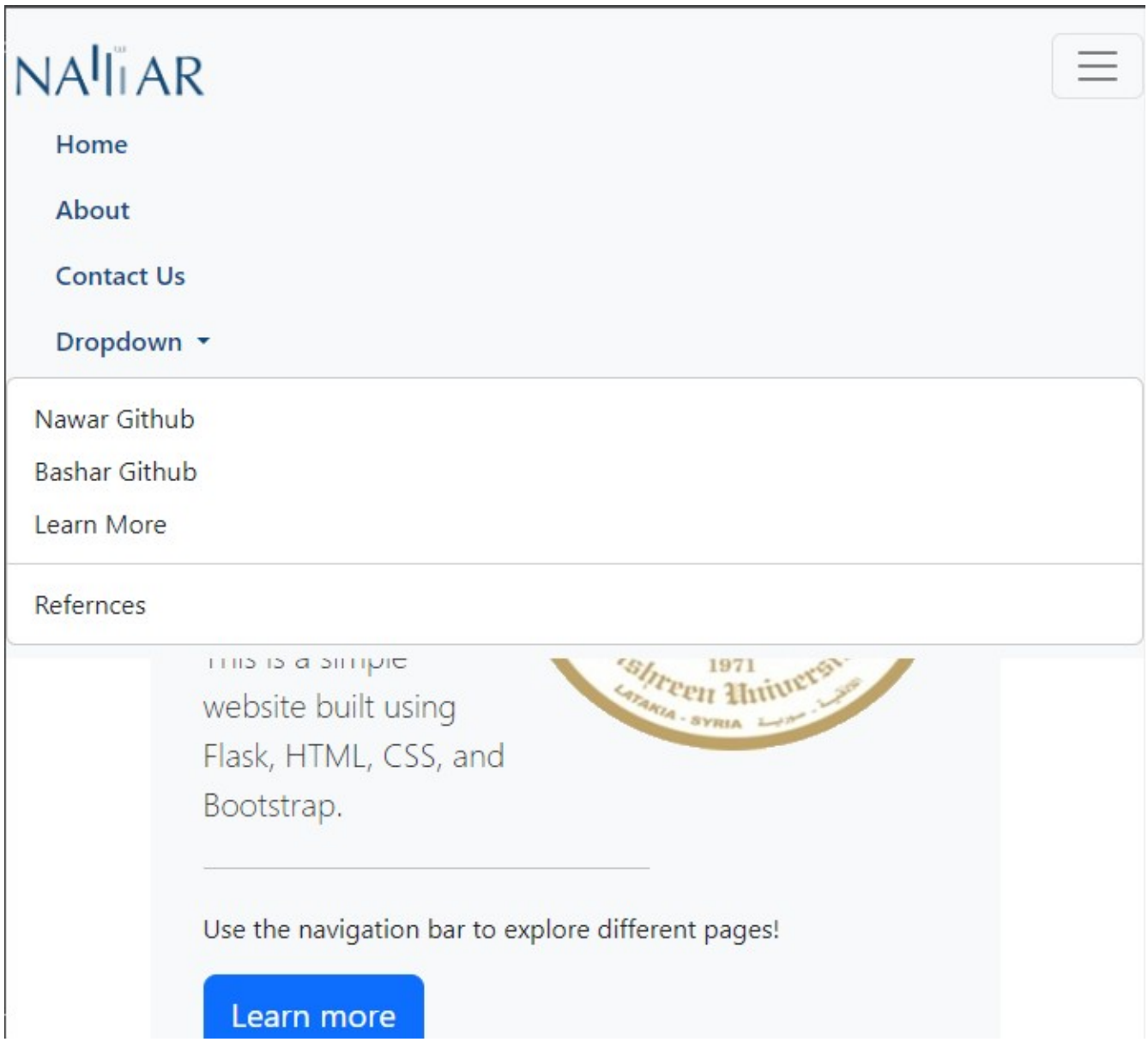
› About page:

› Contact Us page:



» Test if the website is dynamic, responsive and displays correctly on different screen sizes:





ملاحظة: تم ارفاق ملفات ال html وال css الموافقة للصفحات السابقة على github، عبر الرابط التالي:
https://github.com/Nawar095/Network_programming_homework2/tree/main/Question_2