

Syrian Arab Republic

Lattakia – Tishreen
University

Department of
Communication and
electrical engineering

5th, Network
Programming :
Homework No2



الجمهورية العربية السورية

اللاذقية – جامعة تشرين

كلية الهندسة الكهربائية والميكانيكية

قسم هندسة الاتصالات والالكترونيات

السنة الخامسة : وظيفة 2 برمجة

شبكات

Second Network Programming Homework

Students name and number :

سوزان عماد الدين شكوحي 2228

سوسن محمد نور زير 2277

سالي محمد علي رزق 2466

Submitted to GitHub :

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 operation (such as check balance , deposit , and withdraw), And receive their updated completion .

Requirement :

- 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.

```

import socket

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

    while True:
        response = client.recv(1024).decode()
        if not response:
            break

        print(response, end="")

        if "Goodbye!" in response:
            break

        user_input = input()
        client.send(user_input.encode())

if __name__ == "__main__":
    start_client()

```

- هذا الكود يوضح كيفية إنشاء برنامج عميل (client) للتواصل مع خادم (server) عبر شبكة TCP/IP .

تعريفات:

- **socket** : استيراد المكتبة المسؤولة عن إنشاء منافذ للتواصل عبر الشبكة.
- **start_client()** : هي دالة تقوم بتشغيل العميل.

** شرح الدالة ** : start_client()

- إنشاء منفذ اتصال: (socket) :
 - يتم إنشاء منفذ اتصال من نوع socket.AF_INET (عائلة عناوين الانترنت و IPv4) socket.SOCK_STREAM بروتوكول (TCP) ويتم تخزينه في المتغير client (.
- اتصال بالخادم:
 - تتصل الدالة بالخادم على العنوان 127.0.0.1 (العنوان المحلي للجهاز) والمنفذ 9999 .

C. استقبال واستعراض الرسائل:

- i. تدخل الدالة إلى حلقة لا نهائية `while True` حيث:
 - A. يتم استقبال رسالة من الخادم بحجم يصل إلى 1024 Byte وتحويلها إلى نص باستخدام `decode()` وتخزينها في المتغير `response`.
 - B. إذا كانت الرسالة فارغة , (`not response`) فهذا يعني أن الخادم قد أغلق الاتصال، لذا يتم الخروج من الحلقة.
 - C. خلاف ذلك، يتم طباعة الرسالة على الشاشة باستخدام `print`.
 - D. يتم البحث عن عبارة "`Goodbye!`" داخل الرسالة، فإذا وجدت فهذا يعني أن الخادم يطلب إنهاء الاتصال، ويتم الخروج من الحلقة.

d. إرسال رسالة إلى الخادم:

- i. يقوم البرنامج بطلب إدخال نص من المستخدم ويخزنه في المتغير `user_input`.
- ii. يتم تحويل النص إلى بايتات باستخدام `encode()` وإرسالها إلى الخادم باستخدام `client.send`.

تشغيل العميل:

- يتم التحقق فيما إذا كان الكود ضمن الرئيسي : (`if __name__ == "__main__"`)
- إذا كان كذلك، يتم استدعاء دالة `start_client()` لتشغيل العميل.

ملخص:

- يقوم هذا الكود بإنشاء عميل يتصل بخادم على نفس الجهاز (`localhost`) والمنفذ `9999`.
- يقوم العميل باستقبال واستعراض الرسائل التي يرسلها الخادم.
- يمكن للعميل أيضاً إرسال رسائل إلى الخادم عن طريق إدخال نص من لوحة المفاتيح.
- يستمر الاتصال بين العميل والخادم حتى يرسل الخادم رسالة إنهاء "`Goodbye!`".

```

import socket
import threading

# Pre-defined bank accounts
accounts = {
    '12345': {'pin': '1111', 'balance': 1000},
    '67890': {'pin': '2222', 'balance': 1500},
    '54321': {'pin': '3333', 'balance': 500},
}

lock = threading.Lock()

def handle_client(client_socket):
    try:
        # Authenticate client
        client_socket.send(b"Enter account number: ")
        account_number = client_socket.recv(1024).decode().strip()

        client_socket.send(b"Enter PIN: ")
        pin = client_socket.recv(1024).decode().strip()

        if account_number in accounts and accounts[account_number]['pin'] == pin:
            client_socket.send(b"Authentication successful!\n")
        else:
            client_socket.send(b"Authentication failed!\n")
            client_socket.close()
            return

        # Handle client operations
        while True:
            client_socket.send(b"Choose operation: 1. Check Balance 2. Deposit 3. Withdraw 4. Exit\n")
            operation = client_socket.recv(1024).decode().strip()

            if operation == '1':
                balance = accounts[account_number]['balance']
                client_socket.send(f"Your balance is: ${balance}\n".encode())

            elif operation == '2':
                client_socket.send(b"Enter amount to deposit: ")
                amount = float(client_socket.recv(1024).decode().strip())

            with lock:
                accounts[account_number]['balance'] += amount
    except:
        client_socket.close()

```

```

        client_socket.send(b"Deposit successful!\n")

    elif operation == '3':
        client_socket.send(b"Enter amount to withdraw: ")
        amount = float(client_socket.recv(1024).decode().strip())

    )

    with lock:
        if amount <= accounts[account_number]['balance']:
            accounts[account_number]['balance'] -= amount
            client_socket.send(b"Withdrawal successful!\n")
        else:
            client_socket.send(b"Insufficient funds!\n")

    elif operation == '4':
        final_balance = accounts[account_number]['balance']
        client_socket.send(f"Final balance: ${final_balance}\n".
encode())

        client_socket.send(b"Goodbye!\n")
        client_socket.close()
        break

    else:
        client_socket.send(b"Invalid operation!\n")

except Exception as e:
    print(f"Exception: {e}")
    client_socket.close()

def start_server():
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind(('0.0.0.0', 9999))
    server.listen(5)
    print("Server started on port 9999")

    while True:
        client_socket, addr = server.accept()
        print(f"Accepted connection from {addr}")

        client_handler = threading.Thread(target=handle_client, args=(cl
ient_socket,))
        client_handler.start()

if __name__ == "__main__":
    start_server()

```

هذا الكود مكتوب بلغة Python وهو يوضح كيفية إنشاء خادم بسيط لإدارة حسابات بنكية.

المكتبات:

- **socket** : استيراد المكتبة المسؤولة عن إنشاء منافذ للتواصل عبر الشبكة.
- **threading** : استيراد مكتبة خاصة بإنشاء الخيوط (threads) لتحسين أداء الخادم.

المتغيرات:

- **accounts** : قاموس (dictionary) يحفظ بيانات حسابات البنك، حيث:
 - المفتاح : يمثل رقم الحساب البنكي.
 - القيمة : قاموس آخر يحوي معلومات عن الحساب، بما في ذلك:
 - **pin** : الرقم السري للحساب.
 - **balance** : رصيد الحساب.
- **lock** : قفل خاص بمكتبة الخيوط، يُستخدم لمنع حدوث تعارض عند تعديل بيانات الحسابات من قبل خيوط متعددة.

الدوال:

- (^١) **handle_client(client_socket)** : وهي دالة منفصلة تُشغل بخيط خاص لمعالجة كل عميل يتصل بالخادم.
- (a) **عملية المصادقة:**
- i يطلب الخادم من العميل إدخال رقم الحساب والرقم السري.
 - ii يتحقق الخادم من صحة المعلومات عن طريق مطابقتها مع البيانات المخزنة في القاموس **accounts** .
 - iii في حالة نجاح المصادقة، يرسل الخادم رسالة تفيد بذلك.
 - iv في حالة فشل المصادقة، يُغلق الخادم الاتصال مع العميل.

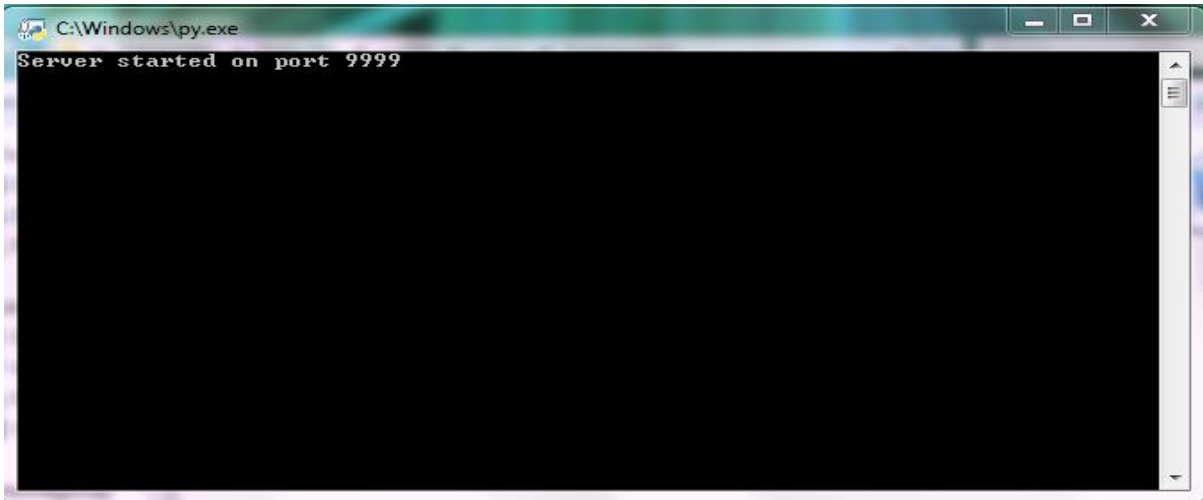
(b) التعامل مع عمليات العميل:

- (i) بعد نجاح المصادقة، يعرض الخادم للعميل قائمة بالعمليات المتاحة (استعراض الرصيد، الإيداع، السحب، الخروج).
 - (ii) يختار العميل العملية المطلوبة ويقوم بإرسالها للخادم.
 - (iii) بناءً على العملية المختارة، يقوم الخادم بالوظائف التالية:
 - (١) **Check balance استعراض الرصيد** : يقوم الخادم بإرسال قيمة الرصيد الحالي للعميل.
 - (٢) **Dipost الإيداع** : يطلب الخادم من العميل إدخال المبلغ المراد إيداعه، ويقوم بتحديث رصيد الحساب بعد التأكد باستخدام قفل lock .
 - (٣) **Withdraw السحب** : يطلب الخادم من العميل إدخال المبلغ المراد سحبه، ويتحقق من توفر الرصيد الكافي قبل السحب، ويقوم بتحديث الرصيد باستخدام قفل lock .
 - (٤) **exit الخروج** : يعرض الخادم الرصيد النهائي للعميل ويقوم بإغلاق الاتصال.
 - (iv) في حالة اختيار عملية غير صحيحة، يرسل الخادم رسالة تفيد بذلك.
- (٢) **start_server()** : وهي الدالة الرئيسية التي تقوم بتشغيل الخادم.
- (a) يقوم الخادم بإنشاء منفذ اتصال على المنفذ 9999 وينتظر طلبات الاتصال من العملاء.
- (b) عند اتصال عميل جديد، يقوم الخادم بإنشاء خيط جديد لمعالجة هذا العميل بشكل منفصل، مما يسمح للخادم باستقبال عملاء آخرين في نفس الوقت.

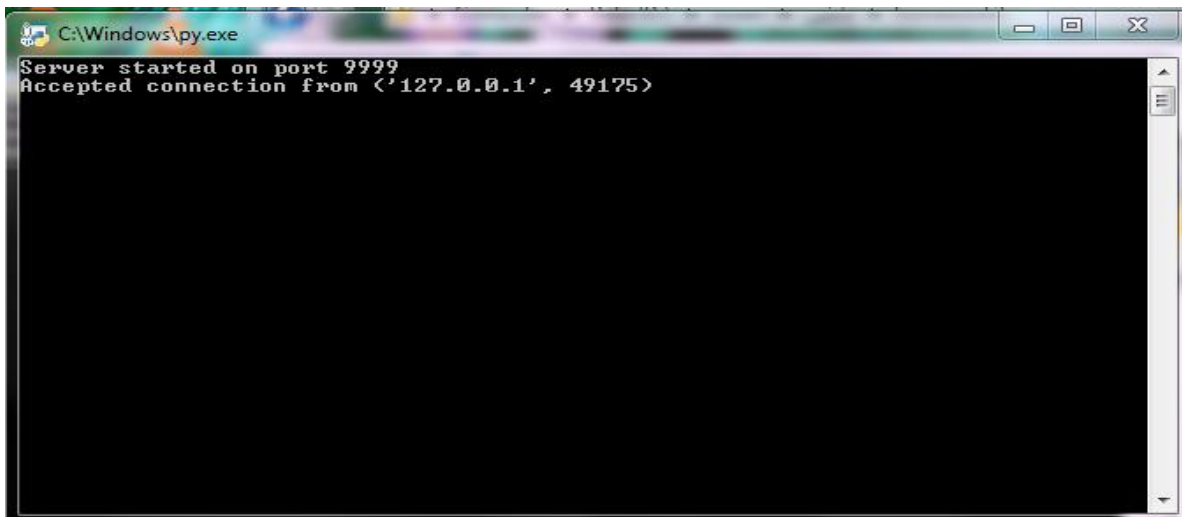
ملخص:

- يوفر هذا الكود خادمًا بسيطًا لإدارة حسابات بنكية، حيث يمكن للعملاء إجراء عمليات استعراض الرصيد والإيداع والسحب بعد المصادقة باستخدام رقم الحساب والرقم السري.
- يستخدم الخادم خيوطًا لتحسين الأداء، بحيث يمكنه التعامل مع عدة عملاء في نفس الوقت.
- من المهم ملاحظة أن هذا مثال بسيط لأغراض التعليم فقط، ولا يوفر جميع الإجراءات الأمنية اللازمة لخادم بنكي حقيقي.

الخرج :

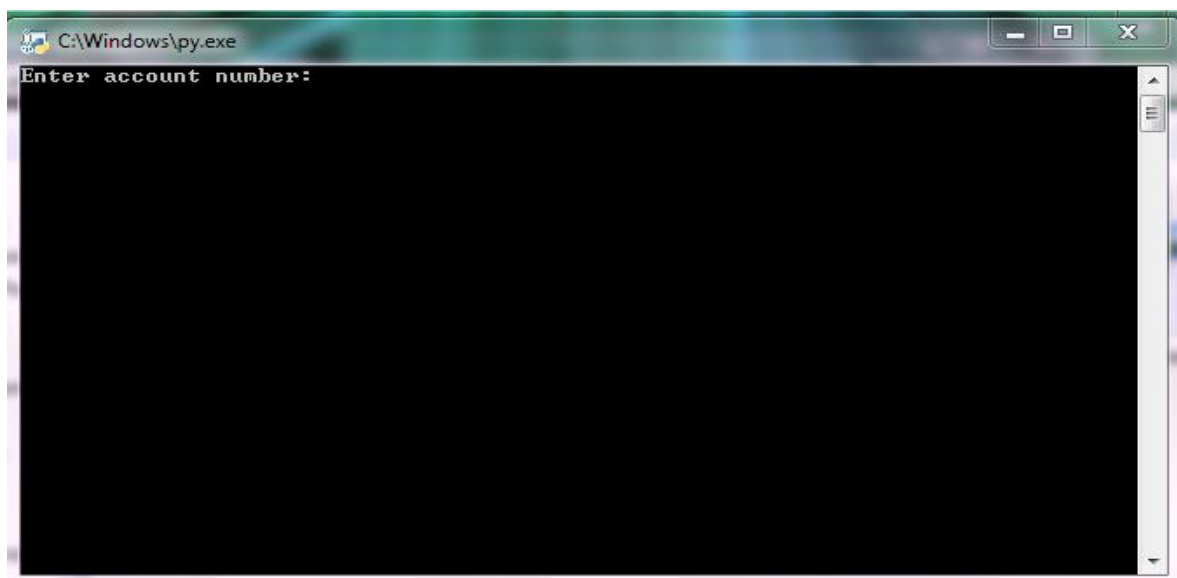


```
C:\Windows\py.exe
Server started on port 9999
```



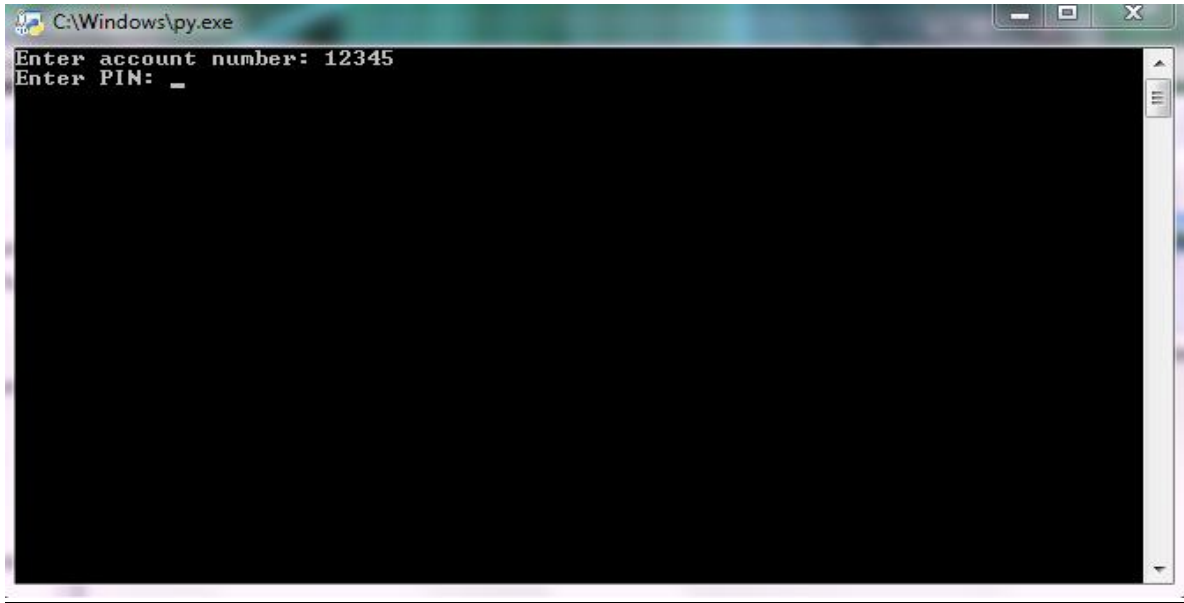
```
C:\Windows\py.exe
Server started on port 9999
Accepted connection from (<'127.0.0.1', 49175>)
```

❖ توضيح الاتصال بالخادم على العنوان 127.0.0.1 (العنوان المحلي للجهاز) والمنفذ 9999 .



```
C:\Windows\py.exe
Enter account number:
```

❖ الطلب من ال client ادخال رقم الحساب .



❖ الطلب من client ادخال رمز ال PIN الخاص بحسابه .



❖ التأكيد على صحة المصادقة أي انه تم الدخول للحساب .

```
C:\Windows\py.exe
Enter account number: 12345
Enter PIN: 1111
Authentication successful!
Choose operation: 1. Check Balance 2. Deposit 3. Withdraw 4. Exit
```

❖ تظهر العمليات التي يمكن client القيام بها من

(check balance , deposit , withdraw) من خلال اختيار الرقم المناسب لكل عملية .

```
C:\Windows\py.exe
Enter account number: 12345
Enter PIN: 1111
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
2
Enter amount to deposit: 200
Deposit successful!

Choose operation: 1. Check Balance 2. Deposit 3. Withdraw 4. Exit
1
Your balance is: $1200.0

Choose operation: 1. Check Balance 2. Deposit 3. Withdraw 4. Exit
3
Enter amount to withdraw: 400
Withdrawal successful!

Choose operation: 1. Check Balance 2. Deposit 3. Withdraw 4. Exit
1
Your balance is: $800.0

Choose operation: 1. Check Balance 2. Deposit 3. Withdraw 4. Exit
```

❖ يظهر الخرج قيام ال client الذي رقم حسابه (12345) ورقم PIN (1111) بالتحقق من رصيد حسابه (\$1000) من ثم قام بإيداع مبلغ ما (\$200) و أعاد التحقق من رصيد

حسابه (1200\$) ثم بعد ذلك قام بسحب مبلغ ما من حسابه (400\$) وتحقق مرة أخرى من رصيده مرة أخرى (800\$).

Question 2 : Simple Wireless 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 .

Requirement :

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 .

```

from flask import Flask, render_template, request, redirect, url_for

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')

# Example of handling form submission
@app.route('/submit', methods=['POST'])
def submit():
    if request.method == 'POST':
        name = request.form['name']
        email = request.form['email']
        message = request.form['message']

        # Process the data (e.g., store in database, send email, etc
        .)

        # For demonstration purposes, we'll just print the data
        print(f"Received message from {name} ({email}): {message}")

        # Redirect to a thank you page after submission
        return redirect(url_for('thank_you'))

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

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

```

هذا الكود يوضح مثلاً بسيطاً لتطبيق Flask يتكون من عدة صفحات ووظيفة لمعالجة إرسال الاستمارة. يعتمد التطبيق على استيراد مكتبات Flask الأساسية لعرض قوالب HTML ومعالجة طلبات HTTP الواردة من المتصفح.

استيراد المكتبات الأساسية (import):

يبدأ الكود باستيراد عدد من المكتبات الأساسية التي يحتاجها التطبيق للعمل. وهي:

- **Flask**: المكتبة الأساسية لبناء تطبيقات الويب باستخدام Python framework Flask.
- **render_template**: دالة من مكتبة Flask تستخدم لعرض قوالب HTML مع بيانات ديناميكية.
- **request**: دالة من مكتبة Flask تستخدم لاستقبال طلبات HTTP الواردة من المتصفح.
- **redirect**: دالة من مكتبة Flask تستخدم لإعادة توجيه المستخدم إلى صفحة أخرى.
- **url_for**: دالة من مكتبة Flask تستخدم لإنشاء روابط (URLs) لصفحات أخرى داخل التطبيق.

إنشاء تطبيق (app):

يتم إنشاء تطبيق Flask جديد باستخدام السطر التالي:

```
app = Flask(__name__)
```

المتغير `__name__` هو متغير خاص في Python يشير إلى اسم الملف الحالي الذي يتم تشغيله.

تحديد مسارات الصفحات (routes):

يستخدم الكود الديكور `@app.route` لتحديد مسارات الصفحات المختلفة في تطبيق الويب. على سبيل المثال:

- `@app.route('/')`: يحدد هذا المسار لصفحة الرئيسية للموقع (عند زيارة الرابط /).
- `@app.route('/about')`: يحدد هذا المسار لصفحة "حولنا" (عند زيارة الرابط /about).
- `@app.route('/contact')`: يحدد هذا المسار لصفحة "اتصل بنا" (عند زيارة الرابط /contact).

وظائف عرض الصفحات (functions):

بعد تحديد مسارات الصفحات، يتم تعريف وظائف منفصلة لكل صفحة. على سبيل المثال:

- `def index()`: هذه الوظيفة مسؤولة عن عرض صفحة الرئيسية (index.html).
 - `def about()`: هذه الوظيفة مسؤولة عن عرض صفحة "حولنا" (about.html).
 - `def contact()`: هذه الوظيفة مسؤولة عن عرض صفحة "اتصل بنا" (contact.html).
- كل وظيفة من هذه الوظائف تستخدم دالة `render_template` لعرض قالب HTML المقابل لها.

معالجة إرسال الاستمارات (form submission):

يوضح الكود مثلاً على معالجة إرسال الاستمارة في صفحة معينة (مثل صفحة اتصل بنا). يستخدم السطر التالي لتحديد المسار الذي سيتعامل مع طلبات POST (إرسال الاستمارة):

Python

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

داخل وظيفة `submit ()` :

- يتم التحقق من طريقة الطلب (`request.method`) للتأكد من أنها POST (إرسال الاستمارة).
- يتم استخراج بيانات الاستمارة مثل الاسم (`name`) والبريد الإلكتروني (`email`) والرسالة (`message`) باستخدام خاصية `request.form`.
- في هذا المثال، يتم طباعة البيانات لأغراض العرض (يمكن استبدالها بتخزين البيانات في قاعدة بيانات أو إرسالها بالبريد الإلكتروني).
- يتم إعادة توجيه المستخدم إلى صفحة شكر بعد إرسال الاستمارة باستخدام دالة `redirect` و `url_for('thank_you')`.

صفحة الشكر (thank you):

تُعرف وظيفة `thank_you ()` لعرض صفحة شكر بسيطة (`thank_you.html`) بعد نجاح إرسال الاستمارة.

تشغيل التطبيق (if __name__ == "__main__"):

الكتلة الأخيرة من التعليمات محاطة بشرط `__main__ == __name__` : . هذه الكتلة يتم تنفيذها فقط عند تشغيل الملف مباشرة (وليس عند استيراده كمكتبة). يقوم هذا الجزء بتشغيل تطبيق Flask في وضع التصحيح (`debug=True`)، مما يسمح بإعادة تحميل التطبيق تلقائياً عند إجراء أي تغييرات في أكواد Python.

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Simple Website</title>
  <link href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/
bootstrap.min.css" rel="stylesheet">
</head>
<body>
  <nav class="navbar navbar-expand-lg navbar-light bg-light">
    <a class="navbar-brand" href="/">My Website</a>
    <button class="navbar-toggler" type="button" data-
toggle="collapse" data-target="#navbarNav" aria-
controls="navbarNav" aria-expanded="false" aria-
label="Toggle navigation">
      <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>
        <li class="nav-item">
          <a class="nav-link" href="/contact">Contact</a>
        </li>
      </ul>
    </div>
  </nav>

  <div class="container">
    <div class="jumbotron">
      <h1 class="display-4">Welcome to My Website</h1>
      <p class="lead">This is a simple website built with Flas
k, HTML, CSS, and Bootstrap.</p>
      <hr class="my-4">
      <p>Use the navigation bar to explore the website.</p>
    </div>
  </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/d
ist/umd/popper.min.js"></script>

```



```
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</body>
</html>
```

هذا الكود يمثل هيكل صفحة ويب بسيطة تستخدم مكتبة Bootstrap لإنشاء عناصر واجهة مستخدم جاهزة. تتضمن الصفحة شريط تنقل لعرض روابط الصفحات الرئيسية ("Home") و "About" و "Contact" بالإضافة إلى مقدمة تعريفية بالموقع.

اعلان نوع المستند (DOCTYPE):

<!DOCTYPE html>: يحدد هذا السطر نوع المستند بأنه صفحة HTML.

بنية الصفحة (html):

(١) **<"html lang="en">**: يمثل العنصر الأساسي للصفحة ويعرف اللغة المستخدمة (en للإنجليزية في هذه الحالة).

(٢) **<head>**: يضم معلومات تعريف الصفحة التي لا تظهر للمستخدم بشكل مباشر، مثل العنوان والترميز والروابط الخارجية.

(a) **<"meta charset="UTF-8">**: يحدد ترميز النصوص المستخدم وهو UTF-8 الذي يدعم معظم اللغات.

(b) **<"meta name="viewport" content="width=device-width, initial-scale=1.0">**: يضبط عرض الصفحة ليستجيب لمختلف أبعاد الشاشات.

(c) **<title/>Simple Website<title>**: يحدد عنوان الصفحة الذي يظهر في شريط المتصفح.

(d) **<link href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet">**

يضيف رابطاً لملف نمط خارجي (stylesheet) وهو مكتبة Bootstrap التي توفر عناصر واجهة مستخدم جاهزة للاستخدام.

(٣) **<body>**: يضم محتوى الصفحة الذي يراه المستخدم.

(a) **<"nav class="navbar navbar-expand-lg navbar-light bg-light">**: يمثل شريط التنقل (navigation bar) الذي يحتوي على روابط الصفحات.

(i) **My Website**: يمثل شعار الموقع (brand) الذي يحتوي على اسم الموقع ورابط الصفحة الرئيسية (/).

(ii) **<"button class="navbar-toggler">** ... : يمثل زر التحكم بقائمة التنقل في الشاشات الصغيرة.

(iii) `<div class="collapse navbar-collapse" id="navbarNav">` يضم قائمة التنقل التي تظهر عند الضغط على زر التحكم.

(١) `<ul class="navbar-nav">` قائمة غير مرتبة (unordered list) تحتوي على عناصر التنقل.

(a) `<li class="nav-item">` كل عنصر (li) يمثل خيارًا في قائمة التنقل.

(i) `Home` يمثل رابط (link) إلى الصفحة الرئيسية (/) مع النص المعروض "Home".

(iv) عناصر أخرى ضمن شريط التنقل تحدد خيارات "About" و "Contact" بنفس الطريقة.

(b) `<div class="container">` يضيف حاوية (container) لتنظيم محتوى الصفحة الأساسي.

(i) `<div class="jumbotron">` يضيف صندوق مميز (jumbotron) لعرض مقدمة الموقع.

(١) `<h1>Welcome to My Website</h1>` يضيف عنوان رئيسي (h1) لعرض عنوان الموقع.

(٢) `<p class="lead">... This is a simple website</p>` يضيف فقرة (paragraph) توضيحية.

(٣) `<hr class="my-4">` يضيف فاصل أفقي.

(٤) `<p>... Use the navigation bar to explore</p>` فقرة إضافية تشجع المستخدم على استكشاف الموقع.

(٤) `<script>` عناصر script تستخدم لتحميل وتحميل مكتبات جافاسكريبت الخارجية.

(a) ثلاثة روابط لمكتبات Bootstrap الخارجية (jQuery و js.Popper) وهي ضرورية لتشغيل وظائف Bootstrap بشكل صحيح.

```
{% block content %}
<div class="jumbotron">
  <h1 class="display-4">Welcome to My Website</h1>
  <p class="lead">This is a simple website built with Flask, HTML,
  CSS, and Bootstrap.</p>
  <hr class="my-4">
  <p>Use the navigation bar to explore the website.</p>
</div>
{% endblock %}
```

قالب Jinja2 - كتلة المحتوى (content):

- هذا الكود يمثل جزء من قالب Jinja2 يُستخدم عادةً مع تطبيق Flask لبناء مواقع الويب. تحديداً، يركز هذا الجزء على تعريف كتلة محتوى قابلة لإعادة الاستخدام.
- `{% block content %}`: يحدد هذا السطر بداية كتلة تحمل اسم "content" (محتوى) داخل قالب Jinja2. تسمح لك هذه الكتل بتعريف محتوى قابل لإعادة الاستخدام في قوالب أخرى.

محتوى كتلة "content" (محتوى):

- `<div class="jumbotron">`: هذا العنصر يُنشئ قسم مميز (jumbotron) وهو منطقة بارزة تستخدم عادةً لعرض مقدمة أو رسالة رئيسية في الصفحة.
- `<h1/>Welcome to My Website<h1>`: يضيف عنوان رئيسي (h1) بنص "Welcome to My Website" (مرحباً بكم في موقعي).
- `<p class="lead">... This is a simple website</p>`: يضيف فقرة (paragraph) تحتوي على نص تعريفى بسيط عن الموقع.
- `<hr class="my-4">`: يضيف خط أفقي للفصل البصري.
- `<p/>... Use the navigation bar to explore<p>`: يضيف فقرة إضافية تشجع المستخدم على استكشاف الموقع من خلال شريط التنقل.
- `{% endblock %}`: يحدد هذا السطر نهاية كتلة المحتوى "content".

استخدام الكتلة في Flask (فلاسك):

- من المرجح أن يُستخدم هذا الكود ضمن ملف قالب Flask (مثل index.html). يمكن استبدال محتوى كتلة "content" (محتوى) في قوالب أخرى موروثة من هذا القالب الأساسي. هذا يتيح لك إنشاء تخطيط متنسق مع رأس صفحة وشريط تنقل وتذييل قابلة لإعادة الاستخدام، مع تخصيص المحتوى الرئيسي لكل صفحة على حدة.
- على سبيل المثال، يمكنك امتلاك قوالب منفصلة لصفحات "عن الموقع" (about.html) و "اتصل بنا" (contact.html) التي ترث من القالب الأساسي وتحدد محتواها الخاص داخل كتلة "content"، مما يؤدي إلى استبدال قسم jumbotron بمحتوى مخصص لكل صفحة.

```

{% extends "base.html" %}

{% block content %}
    <h1>Contact Us</h1>
    <p>This is the contact page. You can put your contact information
here.</p>

    <div class="contact-form">
        <form action="{% url_for('submit') %}" method="post">
            <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="5" required></textarea>
            </div>
            <button type="submit" class="btn btn-primary">Submit</button>
        </form>
    </div>

<style>
    /* General Styles */
    body {
        font-family: Arial, sans-serif;
        background-color: #f5f5f5;
    }

    .contact-form {
        width: 500px;
        margin: 50px auto;
        padding: 20px;
        border: 1px solid #ccc;
        border-radius: 5px;
        box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
    }

    /* Form Styles */
    .form-group {

```

```

    margin-bottom: 20px;
}

.form-control {
    width: 100%;
    padding: 10px;
    border: 1px solid #ccc;
    border-radius: 4px;
}

.form-control:focus {
    outline: none;
    box-shadow: 0 0 0 2px #007bff;
}

label {
    display: block;
    margin-bottom: 5px;
    font-weight: bold;
}

/* Button Styles */
.btn {
    padding: 10px 20px;
    border: none;
    border-radius: 4px;
    cursor: pointer;
}

.btn-primary {
    background-color: #007bff;
    color: #fff;
}

.btn-primary:hover {
    background-color: #0056b3;
}
</style>
{% endblock %}

```

هذا الكود يمثل قالب صفحة اتصل بنا في تطبيق Flask. يستفيد من قالب أساسي "html.base" لتوفير الهيكل المشترك للموقع ويضيف محتوى خاصًا بالصفحة (عنوان، نموذج اتصال، أنماط) داخل كتلة المحتوى المخصصة.

استنساخ قالب أساسي ("extends" base.html) :

- `{% extends "base.html" %}`: يأخذ هذا السطر قالب آخر باسم "html.base" ويستخدمه كأساس للصفحة الحالية. ، يتضمن قالب "html.base" عناصر مشتركة لجميع الصفحات مثل الهيكل الأساسي وشريط التنقل والتذييل.

تعريف محتوى الصفحة (block content) :

- `{% block content %} ... {% endblock %}`: هذه الكتلة تحدد محتوى الصفحة الفريد (صفحة اتصل بنا في هذه الحالة). أي محتوى يوضع بين هاتين علامتين سيتم إدراجه مكان كتلة المحتوى المحددة في القالب الأساسي "html.base".

محتوى صفحة اتصل بنا:

- `<h1/>Contact Us<h1>`: يضيف عنوان رئيسي "اتصل بنا".
- هذا النص هو صفحة الاتصال...: فقرة توضيحية.
- نموذج الاتصال: يضيف قسم نموذج الاتصال (contact-form).
 - نموذج لإرسال اسم (name) والبريد الإلكتروني (email) والرسالة (message) باستخدام طريقة POST إلى مسار `{{ url_for('submit') }}`.

أسلوب (style) CSS:

يحدد هذا القسم أنماط CSS لتنسيق عناصر الصفحة مثل نموذج الاتصال والزر.


```

line-height: 1.6;
font-size: 16px; /* Comfortable body text size */
}

/* Additional styling for responsiveness */
@media (max-width: 768px) {
  .about-us-content {
    width: 95%;
  }
}

@media (max-width: 480px) {
  .about-us-text h2 {
    font-size: 1.8rem;
  }

  .about-us-text p {
    font-size: 14px;
  }
}

.about-us-content {
  /* Existing styles */
  background-
image: url="{{ url_for('static', filename='images.jpg') }}";
  background-size: cover;
  background-position: center; /* Center the image */
}

</style>

{% endblock %}

```

هذا الكود يمثل قالب صفحة about "حولنا" في تطبيق Flask. يستفيد من قالب أساسي "html.base" للبنية المشتركة ويضيف محتوى خاصًا بالصفحة (عنوان، نص تعريف، أنماط) داخل كتلة المحتوى المخصصة. بالإضافة إلى ذلك، تم تضمين صورة خلفية في هذا المثال باستخدام خاصية url_for لإنشاء مسار مناسب للصورة.

استنساخ قالب أساسي ("extends"base.html) :

- `{% "extends"base.html %}` : مشابه للكود السابق، يستخدم هذا السطر قالبًا أساسيًا باسم "html.base" كأساس للصفحة الحالية.

تعريف محتوى الصفحة (block content) :

- `{% block content %} ... {% endblock %}`: هذه الكتلة تحدد محتوى الصفحة الفريد (صفحة حولنا في هذه الحالة).

محتوى صفحة حولنا:

- `<h1/>About Us<h1>`: يضيف عنوان رئيسي "حولنا".
- هذا النص هو صفحة حولنا...: فقرة توضيحية.
- قسم محتوى حولنا (about-us-content):
 - يضم نصًا تعريفياً بالشركة أو الموقع (about-us-text).
 - `<h2/>Who We Are<h2>`: عنوان فرعي "من نحن".
 - فقرات توضيحية حول الشركة أو الموقع.

أسلوب (style) CSS:

- يحدد هذا القسم أنماط CSS لتنسيق عناصر الصفحة:
 - أنماط أساسية للصفحة (خطوط، ألوان الخلفية).
 - تنسيق محتوى حولنا (محاذاة النص، حجم الخطوط).
 - استعلامات media لإستجابة أفضل بأحجام شاشات مختلفة (هواتف محمولة مثلاً).

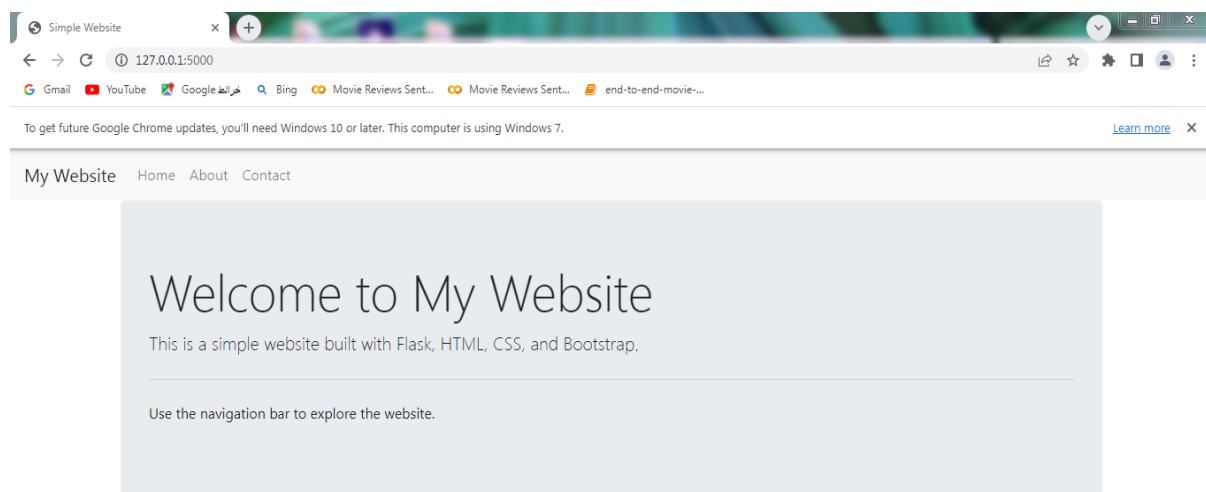
إضافة صورة خلفية (جديد):

- يستخدم `url_for('static', filename='images.jpg')` لإنشاء مسار للصورة الموجودة ضمن مجلد `.static`.
- بإستخدام خصائص أخرى يتم التحكم بمظهر الصورة (تغطية كامل المساحة، محاذاة بالوسط).

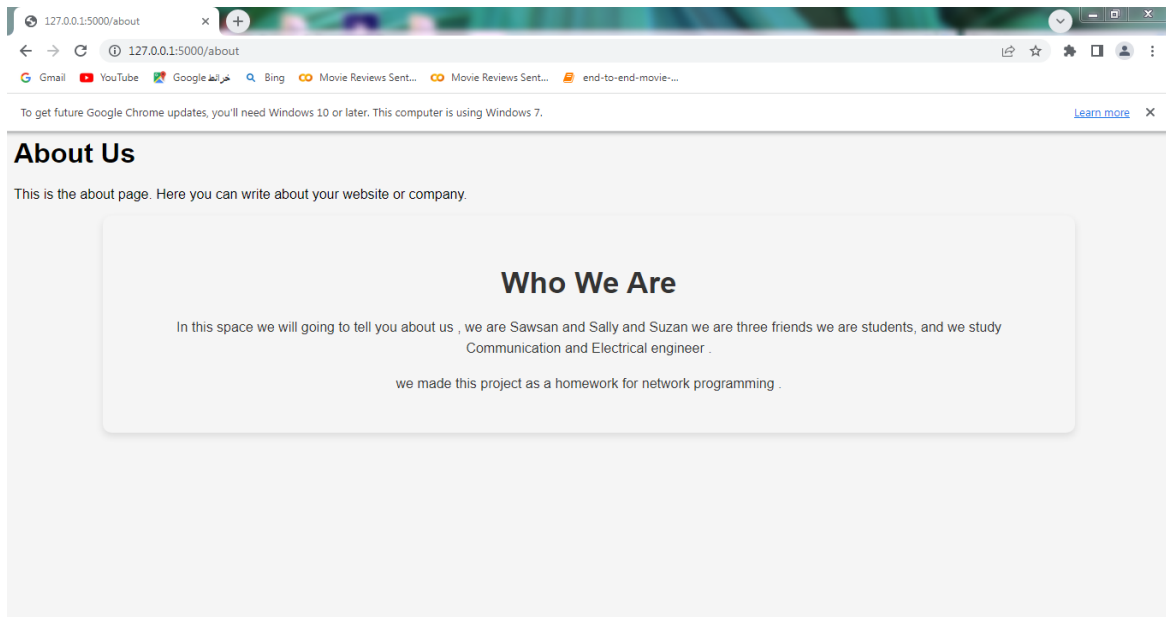
الخروج:

```
C:\Windows\py.exe
* Serving Flask app "app1" <lazy loading>
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Restarting with stat
* Debugger is active!
* Debugger PIN: 895-797-628
* Running on http://127.0.0.1:5000/ <Press CTRL+C to quit>
```

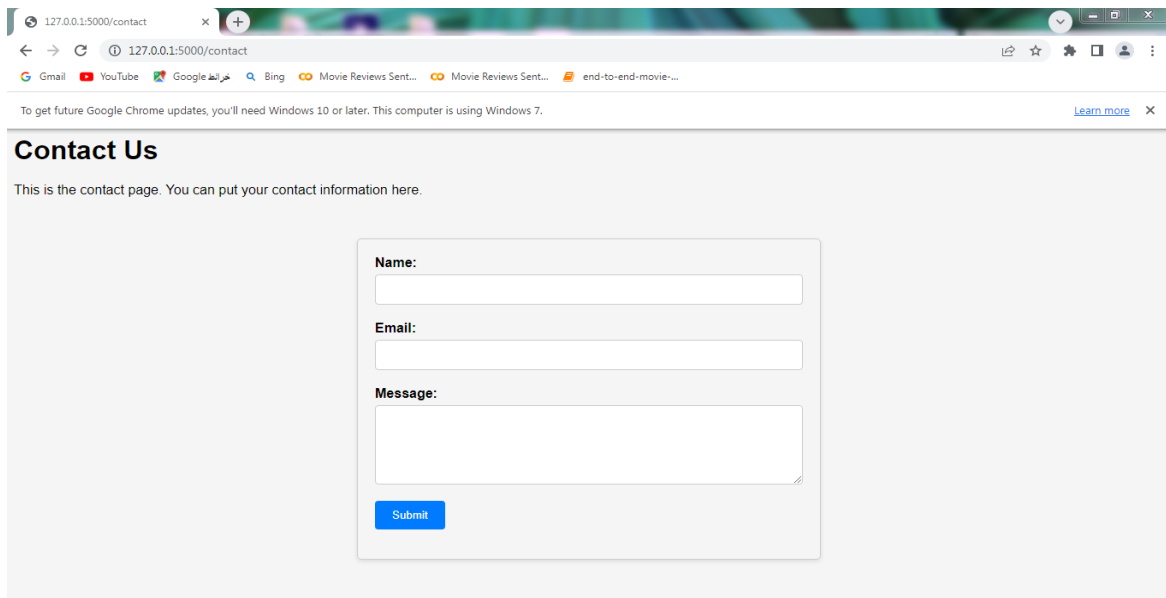
✓ تشغيل السيرفر و توضيح الاتصال بالخادم على العنوان 127.0.0.1 (العنوان المحلي للجهاز) والمنفذ 5000 .



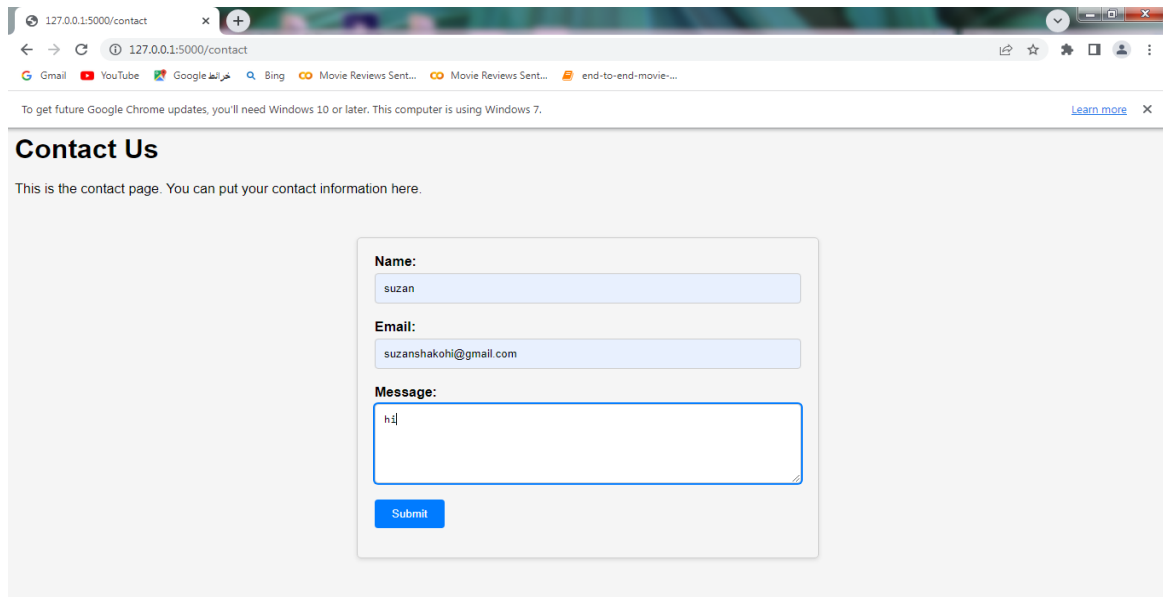
✓ صفحة الويب تم الوصول اليها عبر الرابط <http://127.0.0.1:5000/> .



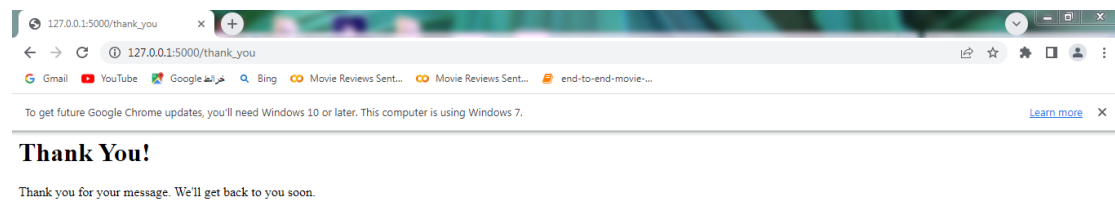
✓ واجهة about يكتب فيها معلومات حول الموقع او الشركة .



✓ واجهة contact us يتم فيها ادخال معلومات المستخدم ورسالته .



✓ تم ادخال معلومات احد المستخدمين .



✓ بعد ادخال المعلومات والضغط على submit تظهر على الواجهة رسالة
thank you!

```
C:\Windows\py.exe
* Serving Flask app "app1" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Restarting with stat
* Debugger is active!
* Debugger PIN: 895-797-628
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [09/Jun/2024 05:23:43] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [09/Jun/2024 05:23:43] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [09/Jun/2024 05:24:21] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [09/Jun/2024 05:24:26] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [09/Jun/2024 05:24:27] "GET /about HTTP/1.1" 200 -
127.0.0.1 - - [09/Jun/2024 05:24:50] "GET /contact HTTP/1.1" 200 -
Received message from suzan <suzanshakohi@gmail.com>: hi
127.0.0.1 - - [09/Jun/2024 05:25:47] "POST /submit HTTP/1.1" 302 -
127.0.0.1 - - [09/Jun/2024 05:25:47] "GET /thank_you HTTP/1.1" 200 -
127.0.0.1 - - [09/Jun/2024 05:26:08] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [09/Jun/2024 05:26:12] "GET / HTTP/1.1" 200 -
```

✓ المعلومات التي قام ال server بمعالجتها وتخزينها بعد الانتهاء من التصفح .

The_End