**Date: 04-06-25**

**Experiment 0**

**Installation of Python Flask for developing web analytics applications and running a basic micro-website to verify the setup.**

**Aim:**

To install Python Flask for developing web analytics applications and to run a basic micro-website to verify the setup.

**Objective**:

Install Flask and necessary tools to create a basic web application with SQLite database support.

**System Requirements**

Operating System: Windows 10/11, macOS (10.14+), or Linux (Ubuntu 20.04+)

RAM: Minimum 4GB (8GB recommended)

Disk Space: 500MB free space

Python Version: 3.7 or higher

**Step-by-Step Setup:**

1. Install Python

Windows/macOS:

Download from python.org

Linux:

sudo apt update && sudo apt install python3 python3-pip

2. Install VS Code (Recommended IDE)

1. Download from [code.visualstudio.com](http://code.visualstudio.com)

2. Install Python extension:

Extensions → Search "Python" → Install

3. Install Flask

pip install flask

4. SQLite Setup

python3 -c "import sqlite3; print(sqlite3.sqlite\_version)"

(Already included with Python)

5. Create Test Application

Save as app.py :

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route('/')

def home():

return "Hello, Flask!"

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

6. Run the Application

Method 1: Terminal

py app.py or python app.py # or python3 app.py

Method 2: VS Code

1. Open app.py

2. Click the ▶️ Run button or press F5

Expected Output:

Terminal: Running on http://127.0.0.1:5000/

Browser: Displays "Hello, Flask!"

Alternative Methods

Virtual Environment

python -m venv myenv

source myenv/bin/activate # Linux/macOS

.\myenv\Scripts\activate # Windows

pip install flask

**Output:**

****

Fig:(/)

**Date: 11-06-25**

**Experiment 1**

**Implement a simple demonstration of creating,retrieving, and deleting Cookies and Sessions**

**Aim:**

To implement a simple demonstration of creating retrieving and deleting cookies and sessions

**Objective:**

Demonstrate Flask's client-side (cookies) and server-side (sessions) state management by creating a web

app that:

Stores/retrieves user data in cookies

Manages user sessions with server-side storage

Clears stored data on demand

**Step-by-Step Setup:**

**1.1 Cookie Implementation**

What we built:

Single HTML page with:

Form to submit username (POST to /setcookie )

Link to check cookie ( /getcookie )

Link to delete cookie ( /clearcookie )

Displays:

Confirmation when cookie is set/deleted

Personalized greeting when cookie exists

Error if no cookie found

**Program**

from flask import Flask, request, make\_response, render\_template\_string

app = Flask(\_\_name\_\_)

# HTML template with a form and links to get and clear cookies.

template = """

<!doctype html>

<title>Cookie Demo</title>

<h2>Cookie Example</h2>

<!-- Simple form to accept user's name -->

<form method="POST" action="/setcookie">

Enter your name: <input type="text" name="username">

<input type="submit" value="Set Cookie">

</form>

<br>

<!-- Links to check and clear cookies -->

<a href="/getcookie">Check Cookie</a> <br>

<a href="/clearcookie">Clear Cookie</a>

"""

@app.route('/')

def home():

# render\_template\_string is used to render inline HTML content

return render\_template\_string(template)

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

def setcookie():

username = request.form.get('username')

# make\_response allows us to attach cookies to a response object

resp = make\_response("Cookie has been set for user: {}<br><a href='/'>Go back</a>".format(username))

if username:

# set\_cookie sets a cookie key-value pair in the response

resp.set\_cookie('username', username)

return resp

@app.route('/getcookie')

def getcookie():

# request.cookies is used to access cookies sent by the browser

username = request.cookies.get('username')

if username:

return f"Hello {username}, welcome back!"

else:

return "No cookie found. Please enter your name first."

@app.route('/clearcookie')

def clearcookie():

# make\_response allows modification of the response before returning

resp = make\_response("Cookie has been cleared!")

# Setting cookie value to empty and expiry to 0 deletes the cookie

resp.set\_cookie('username', '', expires=0)

return resp

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**1.2 Session Implementation**

What we built:

Identical HTML structure to cookie demo for comparison:

Form to submit username (POST to /setsession )

Link to check session ( /getsession )

Link to clear session ( /clearsession )

Displays:

Session confirmation messages

Server-side stored username

Clear visual feedback on session state

**Program**

from flask import Flask, session, request, render\_template\_string

app = Flask(\_\_name\_\_)

app.secret\_key = 'your\_secret\_key\_here' # Required to use sessions securely

# HTML template with a form and links to get and clear session data

template = """

<!doctype html>

<title>Session Demo</title>

<h2>Session Example</h2>

<!-- Simple form to accept user's name -->

<form method="POST" action="/setsession">

Enter your name: <input type="text" name="username">

<input type="submit" value="Set Session">

</form>

<br>

<!-- Links to check and clear session -->

<a href="/getsession">Check Session</a> <br>

<a href="/clearsession">Clear Session</a>

"""

@app.route('/')

def home():

# render\_template\_string is used to embed HTML directly in Python

return render\_template\_string(template)

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

def setsession():

username = request.form.get('username')

# Session data is stored server-side, but a session ID is saved in the user's browser

session['username'] = username

return f"Session has been set for user: {username}<br><a href='/'>Go back</a>"

@app.route('/getsession')

def getsession():

# Accessing session data using session dictionary

username = session.get('username')

if username:

return f"Hello {username}, you are logged in via session!"

else:

return "No session found. Please enter your name first.<br><a href='/'>Go back</a>"

@app.route('/clearsession')

def clearsession():

# session.pop removes 'username' key from session storage

session.pop('username', None)

return "Session has been cleared!<br><a href='/'>Go back</a>"

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**Output:**

**Cookie**

****

Fig; (/)

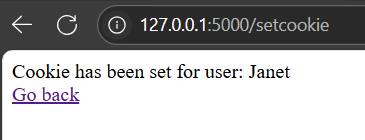
****

Fig:(/setcookie)

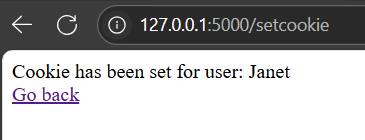
****

Fig:(/getcookie)

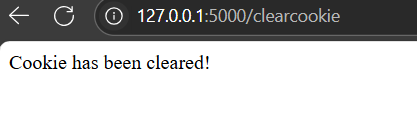
****

Fig:(/clearcookie)

**Session**

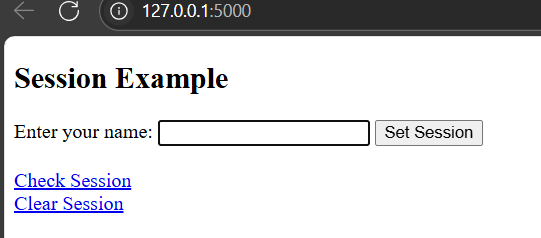
****

Fig:(/)

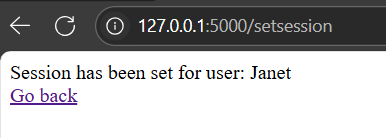
****

Fig:(/setsession)

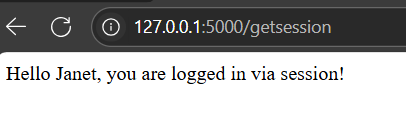
****

Fig:(/getsession)

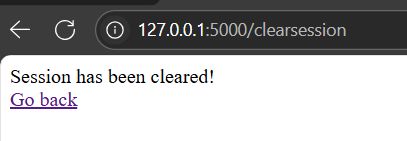
****

Fig:(/clearsession)

**Date: 18-06-25**

**Experiment 2**

**Implement a micro-website with basic functionalities and measure the following two critical web metrics: Conversion Rate and Time on Site**

**Aim:**

To implement a micro-website with basic functionalities and measure the following two critical web metrics: Conversion Rate and Time on Site

**Objective:**

Build a micro-website to track and analyze two critical web metrics:

1. Conversion Rate: Percentage of sessions ending in newsletter signups

2. Time on Site: Duration between session start and end

**Key Components:**

1. Session Tracking:

Starts when visiting /signup

Ends either:

After subscription ( /subscribe route)

On cancellation ( /cancel route)

2. Database Schema:

CREATE TABLE sessions (

id TEXT PRIMARY KEY,

start\_time TEXT,

end\_time TEXT,

converted INTEGER DEFAULT 0

)

3. Metrics Calculated:

# Conversion Rate

(converted\_sessions / total\_sessions) \* 100

# Time on Site

end\_time - start\_time

**Step-by-Step Setup:**

**Program**

from flask import Flask, request, redirect, session

import sqlite3

import uuid

from datetime import datetime

app = Flask(\_\_name\_\_)

app.secret\_key = 'your-very-secret-key-123' # Change this for production

# Initialize database

def init\_db():

conn = sqlite3.connect('sessions.db')

c = conn.cursor()

c.execute('''CREATE TABLE IF NOT EXISTS sessions (

id TEXT PRIMARY KEY,

start\_time TEXT,

end\_time TEXT,

converted INTEGER DEFAULT 0

)''')

conn.commit()

conn.close()

init\_db()

def end\_session(session\_id, converted=False):

"""Helper function to properly end a session"""

if session\_id:

conn = sqlite3.connect('sessions.db')

c = conn.cursor()

c.execute("UPDATE sessions SET end\_time=?, converted=? WHERE id=? AND end\_time IS NULL",

(datetime.now().isoformat(), int(converted), session\_id))

conn.commit()

conn.close()

@app.route('/')

def home():

# Terminate any active session when visiting home

if 'session\_id' in session:

end\_session(session['session\_id'])

session.clear() # Remove all session data

return '''

<h1>Welcome to Our Newsletter</h1>

<p><a href="/signup">Sign Up</a></p>

<p><a href="/metrics">View Metrics</a></p>

'''

@app.route('/signup')

def signup():

# Create new session

session\_id = str(uuid.uuid4())

session['session\_id'] = session\_id

# Store session in DB

conn = sqlite3.connect('sessions.db')

c = conn.cursor()

c.execute("INSERT INTO sessions (id, start\_time) VALUES (?, ?)",

(session\_id, datetime.now().isoformat()))

conn.commit()

conn.close()

return '''

<h1>Newsletter Signup</h1>

<form action="/subscribe" method="POST">

<input type="email" name="email" required placeholder="Your email">

<button type="submit">Subscribe</button>

</form>

<p><a href="/cancel">Cancel Signup</a></p>

'''

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

def subscribe():

if 'session\_id' not in session:

return redirect('/')

email = request.form.get('email', '')

if email:

# Mark session as converted

end\_session(session['session\_id'], converted=True)

session.clear()

return '''

<h1>Thank You!</h1>

<p>You've successfully subscribed.</p>

<p><a href="/">Return Home</a></p>

'''

return redirect('/')

@app.route('/cancel')

def cancel():

if 'session\_id' in session:

end\_session(session['session\_id'])

session.clear()

return redirect('/')

@app.route('/metrics')

def metrics():

conn = sqlite3.connect('sessions.db')

c = conn.cursor()

# Get metrics

c.execute("SELECT COUNT(\*) FROM sessions WHERE end\_time IS NOT NULL")

total\_sessions = c.fetchone()[0]

c.execute("SELECT COUNT(\*) FROM sessions WHERE converted=1")

converted\_sessions = c.fetchone()[0]

conversion\_rate = (converted\_sessions / total\_sessions \* 100) if total\_sessions > 0 else 0

# Get session data

c.execute("SELECT id, start\_time, end\_time FROM sessions ORDER BY start\_time DESC")

sessions = c.fetchall()

conn.close()

# Build sessions table

sessions\_table = []

for s in sessions:

start = datetime.fromisoformat(s[1])

end = datetime.fromisoformat(s[2]) if s[2] else None

duration = str(end - start).split('.')[0] if end else 'N/A'

sessions\_table.append(f'''

<tr>

<td>{s[0][:8]}...</td>

<td>{s[1][11:19]}</td>

<td>{s[2][11:19] if s[2] else 'N/A'}</td>

<td>{duration}</td>

</tr>

''')

return f'''

<style>

table {{ border-collapse: collapse; width: 100%; }}

th, td {{ border: 1px solid #ddd; padding: 8px; text-align: left; }}

tr:nth-child(even) {{ background-color: #f2f2f2; }}

</style>

<h1>Conversion Metrics</h1>

<p>Total Sessions: {total\_sessions}</p>

<p>Converted Sessions: {converted\_sessions}</p>

<p>Conversion Rate: {conversion\_rate:.2f}%</p>

<h2>Session Durations</h2>

<table>

<tr>

<th>Session ID</th>

<th>Start Time</th>

<th>End Time</th>

<th>Duration</th>

</tr>

{"".join(sessions\_table)}

</table>

<p><a href="/">Return Home</a></p>

'''

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**Output:**



Fig:(/)



Fig:(/signup)

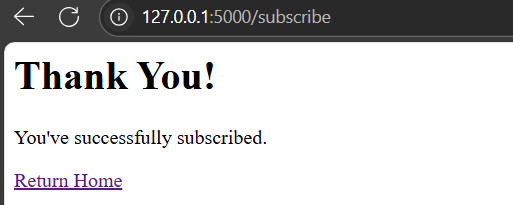


Fig:(/subscribe)

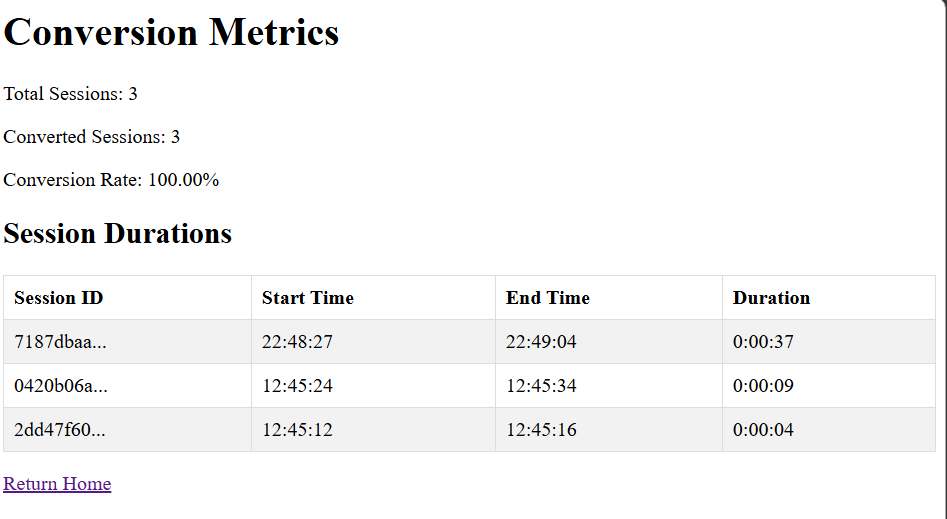


Fig:(/metrics)