CN Lab Assignment 3

Title: Implementation of HTTP Caching and Cookie Management in Python

Part 1: HTTP Caching with ETag and Last-Modified

Aim:

To implement an HTTP server in Python that demonstrates caching using *ETag* and *Last-Modified* headers, and serve web pages efficiently using conditional requests.

Description:

In this part of the assignment, we design a Python-based HTTP server using the `http.server` and `socketserver` modules. The server serves a single `index.html` file, and to reduce redundant data transfers, it implements caching logic by:

- Generating an **ETag** (MD5 hash of file content) for each version of the file.
- Setting the **Last-Modified** header based on the file's modification time.
- Checking the client's request headers *If-None-Match* (ETag) and *If-Modified-Since* to validate cached copies.

If the client already has the latest version of the file:

- The server responds with **304 Not Modified**, avoiding retransmission of content.
  Otherwise:
- The server sends a **200 OK** response along with the file contents, ETag, and Last-Modified headers.

Key Learning Outcomes:

- Understanding how HTTP caching reduces network bandwidth usage.
- Practical demonstration of server-driven caching using both strong (ETag) and weak (Last-Modified) validators.
- Exposure to real-world HTTP header manipulation in server-side programming.

Part 2: Cookie Management with Raw Sockets

Aim:

To implement an HTTP server in Python using raw sockets that manages client sessions using **cookies**.

Description:

This part focuses on handling cookies at the HTTP request-response level without relying on a web framework. The server runs on TCP sockets and implements the following logic:

1. **First-time visitor:**
    - The server sends an HTTP response with `Set-Cookie` header.
    - Assigns a simple value (e.g., "User123") to the cookie.
    - Displays a welcome message for the new user.
2. **Returning visitor (with Cookie):**
    - The server checks if the HTTP request includes a `Cookie` header.
    - Extracts the cookie value and uses it to identify the user.
    - Responds with a personalized "Welcome back" message.

This demonstrates how persistent sessions are maintained between stateless HTTP requests using cookies.

Key Learning Outcomes:

- Understanding of HTTP cookies as a mechanism for client identification and session management.
- Practical demonstration of request parsing and raw HTTP headers using socket programming.
- Realization of statelessness in HTTP and how cookies solve this issue.

Conclusion

- In **Part 1**, we explored **caching mechanisms** using *ETag* and *Last-Modified*, which optimized the communication between client and server.
- In **Part 2**, we implemented **cookie-based user sessions**, enabling the server to recognize returning clients.