Topic 2.3: MCQ Server Project – Part 3

The goal of this project is to set up a basic Python web server that serves multiple-choice questions to the user, scores those questions, and keeps the user's score.

This part of the project allows a user to log on, associating a `username` with each user session.

Goals for Part 3

The goals for this part of the project are:

- Provide a client-side login page retrieved with an HTTP GET request to path `/login`
- Provide a HTTP POST route at path `/login` to receive a login and password from the user and:
  - validate the password based on passwords in a file, `passwords.txt`
  - associate the validated username with the client session cookie on the server
- Redirect clients that are not logged in to the login path (except `/info` and `/logout` routes)

HTTP Status Codes

HTTP status codes are three-digit numbers that a server includes in every response to quickly tell the client what happened. The first digit gives the category:

- `2xx` means success,
- `3xx` is a redirect,
- `4xx` means the client made an error, and
- `5xx` indicates a server problem.

HTTP status code `200` (`OK`) is the standard "everything worked" response. HTTP status `404` (`Not Found`) means the server can't locate the requested resource while `401` (`Unauthorized`) is the response when the resource is found, but the user is not authorized to access it. The most common server-side error is `500` (`Internal Server Error`), a catch-all for unexpected failures of the server. Other codes can be looked up onine.

Our server will use HTTP status code `302` (`Found`) to redirect to the login page if no user is logged in. This status tells the browser to go to a different URL (given in the HTTP `Location` header). This redirection is temporary – the browser should keep requesting the original URL in the future. The HTTP status code `301` (`Moved Permanently`) is used when a resource is permanently moved to a new URL.

New Functions for Handling Authentication: `auth.py`

Add the new file `auth.py` to the project folder. Examine the code for the file. This code has a number of functions relating to authenticating the user.

Function `load_passwords`

The code reads the file `passwords.txt`, where each line is given in the format `username:password`. The code skips blank or malformed lines, and stores the valid pairs in a global dictionary `PASSWORDS` for use in authenticating users. In a production-quality web server, the passwords would be encrypted for an additional level of security. Below is an example `passwords.txt` file:

```
alice:apple
bob:banana
chris:cherry
```

Code Block: Example Passwords File

The function `load_passwords` is called immediately when the server imports the `auth.py` module. Thus, if the `passwords.txt` file is changed, the server must be restarted in order to load the new set of passwords from the file.

Function `parse_post_data`

The function `parse_post_data` reads the request body according to `Content-Length` HTTP header, and parses the URL-encoded form fields into a dictionary. Below shows the example URL-encoded form fields from the example in part 2, along with the resulting Python dictionary produced by the function.

```
username=alice&password=apple
```

```
{username: 'alice', password: 'apple'}
```

Code Block: URL-encoded Example; Resulting Python Dictionary

Function `is_logged_in`

The function `is_logged_in` returns `True` if the session exists and contains a 'user' key (meaning there is a user logged in), otherwise returns `False`.

Function `redirect_to_login`

The function `redirect_to_login` sends a `302` HTTP redirect status code to the login page (the "`/login`" path), optionally adding `?error=1` to trigger an error message on the login page if the user made an unsuccessful login attempt.

Function handle_login

The function handle_login validates the submitted username/password. On success, the function stores the username in the session dictionary and redirects to the client root path (/). If the validation fails, the client is redirected back to /login, with an error in the query string, so: /login?error=1.

```
 1  def handle_login(handler):
 2      """
 3      Process a POST to /login:
 4        - Extract username and password from the form data.
 5        - Check them against PASSWORDS.
 6        - On success:
 7             store 'user' in session,
 8             set session cookie,
 9             redirect to the home page.
10        - On failure: redirect back to /login?error=1
11      """
12      # Safety check: dispatcher should have called get_session()
13      if not hasattr(handler, 'session') or handler.session is None:
14          handler.send_error(500, 'No session object')
15          return
16      data = parse_post_data(handler)
17      username = data.get('username', '').strip()
18      password = data.get('password', '')
19      # Compare against the loaded password file
20      if username in PASSWORDS and PASSWORDS[username] == password:
21          # Successful login – remember the user in the session
22          handler.session['user'] = username
23          handler.send_response(302)
24          handler.send_header('Location', '/')   # redirect to home page
25          session.set_session_cookie(handler)    # send cookie
26          handler.end_headers()
27      else:
28          # Login failed – redirect back to /login with error indicator
29          redirect_to_login(handler, error=True)
```

Code Block: Complete code for function handle_login

Lines 13-15 exit the function if it were erroneously called without an existing session, sending HTTP status code 500 (Internal Server Error) and an error message to the client.

Lines 16-18 calls the local function parse_post_data to get the username and password from the body of the HTTP request message.

Line 20 validates the submitted username and password by consulting the dictionary PASSWORD that was loaded from passwords.txt during server startup. On successful validation, line 22 associates the username with the current session identifier (stored in SESSIONS[sid] from sessions.py), and lines 23-26 send the response message back to the server with the session cookie, and HTTP status code 302, telling the client to load the URL root path (/).

If the `username/password` combination is not valid, line 29 calls the function `redirect_to_login` with the parameter `error` set to `True` so the user will receive the login page again, with an error message displayed.

Function `handle_logout`

The function handle_login removes the `username` from the session (line 8), expires the session cookie and deletes the session identifier from list of sessions (line 13), and redirects the client to the login page (lines 10-11).

```
 1  def handle_logout(handler):
 2      """
 3      Log out the current user: remove 'user' from session,
 4      expire the session cookie, and redirect to login page.
 5      """
 6      # Remove user from session if present
 7      if hasattr(handler, 'session') and handler.session is not None:
 8          handler.session.pop('user', None)
 9      # Start the redirect response
10      handler.send_response(302)
11      handler.send_header('Location', config.LOGIN_ROUTE)
12      # Now expire the session cookie (adds a Set-Cookie header)
13      session.clear_session_cookie(handler)
14      handler.end_headers()
```

Code Block: Complete code for function `handle_logout`

New client-side file: `www/login.html`

The `login.html` file is a new file that should be placed in the `www` directory (the directory for client-side files). The server code specifies the filename exactly, so the server will not work properly without the file.

The page contains a login HTML form (lines 12-22) that contains input `text` fields for the `username` and `password`, as well as a `button` of type `submit`.

Inline JavaScript (lines 24-31) will add a simple error message, "`Invalid username or password.`" onto the page above the form if the server has redirected the client to request the path with the error query string (i.e.: "`/login?error=1`").

```
 1 <!DOCTYPE html>
 2 <html lang="en">
 3 <head>
 4     <meta charset="UTF-8">
 5     <title>Login</title>
 6     <style>
 7         .error { color: red; }
 8     </style>
 9 </head>
10 <body>
11     <h1>Login</h1>
12     <form action="/login" method="post">
13         <label>Username:
14             <input type="text" name="username" required>
15         </label>
16         <br>
17         <label>Password:
18             <input type="password" name="password" required>
19         </label>
20         <br>
21         <button type="submit">Log in</button>
22     </form>
23     <script>
24        // Show error message if redirected with query ?error=1
25        const params = new URLSearchParams(window.location.search);
26        if (params.get('error') === '1') {
27           const p = document.createElement('p');
28           p.className = 'error';
29           p.textContent = 'Invalid username or password.';
30           document.body.insertBefore(p,document.querySelector('form'));
31        }
32     </script>
33 </body>
34 </html>
```

Code Block: Complete code for file `login.html`

Upgrading the server

auth.py, passwords.txt, login.html

Ensure the file `auth.py` and `passwords.txt` are copied into the project root directory, and `login.html` is copied into the www directory.

www/index.html

Update the file `www/index.html` to add a logout button (lines 13-15).

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>MCQ-Server</title>
7  </head>
8  <body>
9      <h1>MCQ Server</h1>
10     <ul>
11         <li><a href="/info">Connection Information Page</a></li>
12     </ul>
13     <form action="/logout" method="get">
14         <button type="submit">Logout</button>
15     </form>
16 </body>
17 </html>
```

Code Block: Complete code for file `index.html`

mcq-server.py

The do_GET method is updated. Each select (if) statement serves a single route. Lines 1-8 and 28-29 remain unchanged. Lines 9-12 serve the /info route. Lines 13-19 serve the /login route. Lines 20-23 handle the /logout. Lines 24-27 will prevent any other route from being served if a user is not logged in; the client will be instructed to redirect to the /login route.

```
 1      def do_GET(self):
 2          print(f"--> Received GET {self.path}")
 3          parsed = urlparse(self.path)
 4          route_path = parsed.path
 5          # Attach session data to the handler.
 6          # If a cookie exists it will use the given session identifier;
 7          # otherwise a new session identifier is created.
 8          session.get_session(self)
 9          # Public info route (no session or login required)
10          if route_path == config.INFO_ROUTE:
11              server_info.handle(self)
12              return
13          # Public login page (maps route /login to file login.html)
14          # Keeps the query string, if present (for failed logins)
15          if route_path == config.LOGIN_ROUTE:
16              self.path = '/login.html' + ('?' + \
17                          parsed.query if parsed.query else '')
18              static_handler.serve(self)
19              return
20          # Logout - always allowed (even if not logged in)
21          if route_path == config.LOGOUT_ROUTE:
22              auth.handle_logout(self)
23              return
24          # If not logged in, redirect to login page
25          if not auth.is_logged_in(self):
26              auth.redirect_to_login(self)
27              return
28          # All remaining routes are private
29          static_handler.serve(self)
```

Code Block: Updates to mcq-server.py method do_GET

A do_POST method needs to be added to receive the HTTPs POST requests that receive the login information in the message body. So an HTTP GET request message to the path "/login" (the value set by config.LOGIN_ROUTE) will be handled by the do_GET method and serve the login.html file, while an HTTP POST request message to the same path will be handled by do_POST method and handle parsing the login information sent to the server. Examine the code for do_POST given below. The first lines of do_POST are the same as do_GET: the URL path is parsed and the session information is extracted from the cookies. Then, lines 6 to 9 will process the login if the POST request was to "/login", or return a client error (404, Not Found) if the POST request was sent to a different URL path. Method do_POST is to be added to the RequestHandler class at the same level as the do_GET method.

```
    def do_POST(self):
        print(f"--> Received POST {self.path}")
        parsed = urlparse(self.path)
        route_path = parsed.path
        session.get_session(self)
        if route_path == config.LOGIN_ROUTE:
            auth.handle_login(self)
        else:
            self.send_error(404, 'POST route not found')
```

Code Block: Updates to `mcq-server.py` method `do_POST`