Topic 2.3: MCQ Server Project – Part 4

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 adds serving of the multiple-choice questions.

Goals for Part 4

The goals for this part of the project are to serve question routes:

- `/question` – serve a list of questions available on the server
- `/question/#` – serve a specific question number, where # is the integer question number.
- `/question/random` – serve one randomly-chosen question from those questions available on the server

Question File Format

Each question file is a static HTML page with a form that submits to the URL path `/answer`. The form includes a hidden `<input name="question" value="<number>">` to identify the question and a set of radio buttons named choice with values like `A`, `B`, `C`, `D`. When submitted, the browser sends a POST request with a URL-encoded body such as `question=1&choice=A`. Later server functionality will be added to parse this response and check if the answer is correct.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Question 1</title>
</head>
<body>
    <h1>Question 1</h1>
    <p>What does HTTP stand for?</p>
    <form action="/answer" method="post">
        <input type="hidden" name="question" value="1">
        <input type="radio" name="choice" value="A" id="a1">
        <label for="a1"> HyperText Transfer Protocol</label><br>
        <input type="radio" name="choice" value="B" id="b1">
        <label for="b1"> High Tech Transfer Protocol</label><br>
        <input type="radio" name="choice" value="C" id="c1">
        <label for="c1"> Hyperlink Text Transfer Protocol</label><br>
        <input type="radio" name="choice" value="D" id="d1">
        <label for="d1"> High-rate Text Transfer Protocol</label><br>
        <button type="submit">Submit</button>
    </form>
</body>
</html>
```

Code Block: Example Question file: `q1.html`

Question files must be named a lowercase `q` followed by the question number, then `.html`. The question files must all live inside the `www/questions/` directory because the `questions.py` module scans this directory for files matching exactly that pattern.

New Functions for Serving Multiple-Choice Questions: `questions.py`

Add the new file `questions.py` to the project folder. Examine the code for the file. This code has a number of functions relating to serving multiple-choice questions.

Function `handle_question`

This is the only function that is meant to be called externally. It is called when the URL path starts with the prefix "`/question`" (or a different path if `config.QUESTION_ROUTE_PREFIX` is changed). This function calls one of the helper methods:

- `_list_page` – to send the client a page with the list of questions available
- `_random_redirect` – to send the client a random question to answer
- `_serve_question_file` – to send the client a particular question number that they have requested.

More explanation of these functions is given below.

Function `_serve_question_file`

The underscore prefix denotes that this function is intended as a private helper function used only within the file. This is the function searches in the `www/questions/` directory to find the question file of the corresponding question number. For example, if question `12` is requested, the function check for the file `www/questions/q12.html`, and if found, serves that file to the client.

Function `_get_question_numbers`

Another private helper function. This function searches the `www/questions/` directory for files starting with a lowercase `q` followed by the question number, ending with `.html`. It generates a list of question numbers that have corresponding question files available in the directory.

Function `_random_redirect`

This private helper function that is used when the client clicks on the link to get a random question. This function calls `_get_question_numbers` to get the list of question numbers available, randomly chooses one of those available question numbers, then uses the HTTP status code `302` to redirect the client to (tell the client to ask the server for) that question number.

Function `_list_page`

This private helper function that generates the HTML for a page displaying the list of questions available. The page also has links to choose a random question, for the home page, and to logout.

Upgrading the server

`questions.py`

Ensure the file `questions.py` is copied into the project root directory.

`www/questions/q1.html`

Create a `questions` directory within the `www` directory and copy the example question file, `q1.html`, into the directory. Later, add your own question files.

`mcq-server.py`

Add the few lines (lines 24-27, below) that will call the `handle_question` function from `questions.py` when a request is made to a path that is prefixed with `/questions`.

```
 1  def do_GET(self):
 2      print(f"--> Received GET {self.path}")
 3      parsed = urlparse(self.path)
 4      route_path = parsed.path
 5      session.get_session(self)
 6      # Public info route (no session or login required)
 7      if route_path == config.INFO_ROUTE:
 8          server_info.handle(self)
 9          return
10      # Public login page
11      if route_path == config.LOGIN_ROUTE:
12          self.path = '/login.html' + ('?' + \
13                      parsed.query if parsed.query else '')
14          static_handler.serve(self)
15          return
16      # Logout
17      if route_path == config.LOGOUT_ROUTE:
18          auth.handle_logout(self)
19          return
20      # All remaining routes require login
21      if not auth.is_logged_in(self):
22          auth.redirect_to_login(self)
23          return
24      # Question routes
25      if route_path.startswith(config.QUESTION_ROUTE_PREFIX):
26          questions.handle_question(self, route_path)
27          return
28      # Private route: serve static file from www/
29      static_handler.serve(self)
```

Code Block: Complete code for function `do_GET`