Topic 2.3: MCQ Server Project – Part 5

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 scoring of the multiple-choice questions and keeping a record of the number of correctly and incorrectly answered questions for each user.

Goals for Part 5

The goals for this part of the project is to:

- process HTTP POST submissions on route `/answers` (or the route configured by `config.ANSWER_ROUTE`)
- parse the multiple-choice answer submissions from HTML question page forms
- check if the submitted answer is correct based on an answer key provided in the file `answers.txt` present in the project root directory
- keep track of the number of correct and incorrect answer submissions from each user
- provide a score page the user can check at the route /score (or the route configured by `config.SCORE_ROUTE`).

Answer File Format

The answer file must be in the project directory and named `answers.txt`. The file is loaded when the server starts, so the server must be restarted to pick up any updates to the file. Each line should consist of an integer question number followed by whitespace, then the answer. The answer is case insensitive: both the answer from `answers.txt` and the answer submitted from the question page HTML form will be converted to upper case and then compared to determine if the submitted answer is correct. Below is an example `answers.txt` file with answers to three questions.

```
1 A
2 C
3 B
```

Code Block: Example Answer Key File: `answer.txt`

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

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

Function `load_answers`

This function is run at server startup. It loads the `answers.txt` file (format described above) into memory.

Function `parse_post_data`

This function parses the body of the HTTP POST request to create a data object so that the server (function `handle_answer`, described below) can easily get the question number and answer submission.

Function `handle_answer`

This function processes the answer submission (HTTP POST requests to `/answer`). It calls `parse_post_data` to get the question number and the answer submitted by the user, compares the submitted answer to the answer that was loaded from `answers.txt`, updates the number of correct or incorrect answers by the user, and sends the HTTP response page containing the results to the user.

Function `show_score`

This function serves the `/score` route by sending a simple HTML page with the number of correct and incorrect submissions made by the user.

Upgrading the server

`scoring.py, answers.txt`

Ensure the file `scoring.py` and `answers.txt` copied into the project root directory.

`mcq-server.py`

Add the few lines (lines 14-17, below) that will call the `handle_answer` function from `scoring.py` when a HTTP POST request is made for to the question response route, `/answer` (`config.ANSWER_ROUTE`).

```
 1      def do_POST(self):
 2          print(f"--> Received POST {self.path}")
 3          parsed = urlparse(self.path)
 4          route_path = parsed.path
 5          session.get_session(self)
 6          # Login is always public
 7          if route_path == config.LOGIN_ROUTE:
 8              auth.handle_login(self)
 9              return
10          # All other POST routes require login
11          if not auth.is_logged_in(self):
12              auth.redirect_to_login(self)
13              return
14          # Answer submission
15          if route_path == config.ANSWER_ROUTE:
16              scoring.handle_answer(self)
17              return
18          self.send_error(404, 'POST route not found')
```

Code Block: Complete code for function `do_POST`

Add the lines (lines 7-10, below) that will call the `show_score` function from `scoring.py` when a request is made for the score page, route `/score` (`config.SCORE_ROUTE`).

```
1  def do_GET(self):
2      # ### SOME CODE NOT SHOWN ###
3      # Question routes
4      if route_path.startswith(config.QUESTION_ROUTE_PREFIX):
5          questions.handle_question(self, route_path)
6          return
7      # Score page
8      if route_path == config.SCORE_ROUTE:
9          scoring.show_score(self)
10         return
11     # Private: serve static file from www/
12     static_handler.serve(self)
```

Code Block: Partial code for function `do_GET`

Note that the score route can be placed before or after the question routes because they don't interfere with each other. Both these routes must be below the login redirection (to only allow access one the user is logged in) and above the static file serving (so the question and score serving cannot be overridden by static files).