Topic 2.6: MCQ Server Project – Part 6: Extending the Server

The goal of the project was 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.

Although you have completed the project functionality, there are many ways the server could be improved or new functionality could be added.

Complete at least **one** improvement from the *Basic Improvement Ideas* section, and **one** improvement from the *Server Enhancement* section.

Basic Improvement Ideas

Some relatively simple improvements can easily be made to the project:

- ***Additional MCQ Questions*** – Only one question was provided in the documentation, but there should really be
- ***CSS Styling*** – this could be inline styling, but it would be better to create a central CSS file.
- ***Change Password*** – implement the ability for the user to change their password by entering their old password and the new password.
- ***Password Hashing*** – it is best practices to store the password as an encrypted hash (in `passwords.txt`) for additional security in case the web site is hacked. This is easy to code, but will take some time to research how it works.
- ***Persistent Scoring*** – currently scores are only stored in memory, so do not survive a server restart. Save student scores to a file so that they will survive a server restart.
- ***Administrator Page*** – create an administrator page, at the path /admin, that reads the SESSIONS variable to show user sessions and their scores. You should also implement a way to ensure only users who have "administrator privileges" can access the administrator page(s).
- ***Error Handling*** – some situations may not give very good feedback to the user or the server administrator as to what is happening with the server. For example, what happens if `passwords.txt` is missing or `answers.txt` is missing? What happens if there is a question that doesn't have a corresponding answer in the `answers.txt`? Consider how the feedback sent back to the user may differ from the feedback printed to the console for the server administrator.

Server Enhancement

These larger feature ideas will require one or more new Python modules. The first two are relatively easy, the third is a bit more involved, and the fourth will take quite a lot of time and effort – and likely a lot of time trying to debug problems.

1. ***Live Chat / Messaging Board***
   Add a page where logged-in users can post messages that appear for everyone.
   How: Store messages in a global list (or a file). The chat page can auto-refresh every few seconds with an HTML `<meta>` refresh or JavaScript `setInterval`.
   Learning: Shared server state, concurrency considerations, simple polling.
2. ***Leaderboard***
   A public page /leaderboard that displays all users' scores (from SESSIONS or a persistent score file).
   Learning: Extracting data from sessions, sorting, and presenting aggregated information.

3. ***Database Storage (SQLite)***
   Replace the text files (`passwords.txt`, `answers.txt`, scores) with an SQLite database.
   How: Use Python's `sqlite3` module to create tables for users, questions, answers, and scores. Rewrite `auth.py`, `questions.py`, and `scoring.py` to query the database.
   Learning: SQL basics, database design, and the advantages of structured storage;
   NOTE: ***database design is an important and sometimes difficult aspect of the course; it would be good to get early experience with database design and coding.***

4. ***WebSockets*** for Real-time Multiplayer Game or Messenger Application:
   The bare HTTP protocol will set up and tear down TCP connections to send a set of data, while WebSockets keeps the TCP connection open between sets of data to be sent, greatly reducing communication latency between server and client. Thus it is essential to upgrade an HTTP connection to a WebSocket connection for any real-time application, such as a game.
   You could convert the MCQ answering into a real-time competitive game where two or more players see the same question at the same time and race to answer. If you wish to implement a different real-time game where two or more players compete head-to-head, WebSockets communications goes between the user computers and the server to maintain the game state.
   How: Upgrade to a WebSocket-capable server (e.g., using `websockets` library). For the quiz game, the server broadcasts the question and collects answers instantly. For any other real-time game, each client communicates with the server to update the game state.
   Learning: WebSockets, event-driven programming, real-time communication.