PR Lab 2

Deadline: 2 weeks; until 04.11.2024;

In this lab, you will implement an HTTP web server with CRUD operations and run it in a Docker container. Additionally, you'll set up a WebSocket interface on a separate port to handle a chat implementation. Tasks 9 and 10 involve creating a separate server that runs over the TCP protocol, introducing you to threading concepts and the porlbmes that come with them.

Grade	Task Description
1	Select a database to use for the project.
2	Design the data model for your database based on the data you scrape from lab-1. You can use either an ORM or raw SQL queries to interact with the database.
3	Implement CRUD (Create, Read, Update, Delete) operations with an HTTP interface. Use libraries or frameworks as needed. For DELETE, PUT, and GET operations, utilize query parameters (such as ID or name) to identify the resources.
4	Add pagination for your resources (e.g., use 'offset=1' and 'limit=5' to control the result set).
5	Implement a route/handler to accept multipart/form-data file uploads. Test this by sending a JSON file. (through either Postman or a script)
6	Implement a Chat Room logic using the WebSocket protocol.[1]
7	Use Docker Compose to run your database inside a Docker container. Similar to the previous .yaml file, you have to find the appropriate database image and learn how to connect to the database server over the network.
8	Write a 'Dockerfile' for your application. Run it with the docker command.
9	Implement a separate TCP server to handle client connections and messages. See details here: [2]
10	Implement a coordination mechanism to manage the order of read and write operations on the shared file. Ensure that all write operations complete before any read operations. Use synchronization mechanism or other approaches to coordinate the execution order between threads.

[1] The chat room should allow clients to join, send/receive messages and leave rooms. In your application, run both the HTTP web server and the WebSocket handler using two separate threads in the same file (this means you will have two separate controllers or API interfaces on different ports).

[2] Process each message/request in individual threads. Interpret the incoming messages as commands (write and read operations).These commands/threads should operate on a file and should sleep randomly from 1 to 7 seconds before execution. Use your creativity to decide what data to write and read from the file. Given that multiple threads will access the same/a common resource, use locking mechanisms (example: using a Mutex) to ensure safe concurrent access(avoiding race condition).