Assignment 2: Client-Server Posting System

Objective: This assignment focuses on building a basic client-server posting system using Node.js, emphasizing stateful application development, asynchronous communication with `fetch`, and code testing.

Technology Stack:

- **Backend:** Node.js
- **Frontend:** HTML, JavaScript (vanilla - no libraries)
- **Communication:** `fetch` API
- **Containerization:** Docker (Dockerfile and/or docker-compose.yml)

Project Structure:

You will submit a single, compressed archive (`.zip`,) containing the following:

- `docker-compose.yml` (and `Dockerfile` if needed) - for containerization.
- `server.js` (or similarly named) - your Node.js server code.
- `posting.html` - your HTML page with embedded JavaScript.
- `report.pdf` - your test report.

Parts:

Part A: Node.js Backend (40 Points)

1. **Create a POST Endpoint:**
    - Develop a Node.js server application (`server.js`) using the `express` and `body-parser` npm modules.
    - Implement a POST endpoint `/postmessage` that accepts two parameters in the request body:
        - `topic` (string): The topic of the post.
        - `data` (string): The content of the post.
    - The endpoint should append the received `topic`, `data`, and a current timestamp (date and time) to a file named `posts.txt`.
    - **File Handling:**
        - If `posts.txt` exists, append the new post information to it.
        - If `posts.txt` does not exist, create it and then add the post information.
2. **Allowed Modules:**

- You are **required** to use `express` and `body-parser` for handling HTTP requests and parsing request bodies.
- You **may use** standard Node.js built-in modules (e.g., `fs` for file system operations, `Date` for timestamps).
- **No other external npm modules are permitted.**

Example `posts.txt` Content:

Topic: My First Post
Data: This is the content of my post.
Timestamp: 2023-10-27 10:00:00

Topic: Another Post
Data: Some more content here.
Timestamp: 2023-10-27 10:15:32

Part B: HTML Frontend (50 Points)

1. **Create `posting.html`:**
    - Develop a single HTML page named `posting.html` that serves as the user interface for your posting system.
    - The page should display all existing posts from the `posts.txt` file.
    - The page should provide a form that allows users to create new posts by entering a topic and data.
    - **Asynchronous Communication:** Use **only** asynchronous `fetch` calls to communicate with your Node.js server.
    - **Real-time Updates:** If the current user adds a new post, the page should refresh the post list without requiring a manual page reload.
    - **No JavaScript Libraries:** You are **not allowed** to use any JavaScript libraries (e.g., jQuery, React, Axios) for this part. All functionality must be implemented using vanilla JavaScript.

Part C: Test Report (10 Points)

1. **Create `report.pdf`:**
    - Write a test report (in PDF format) that documents how you tested your Node.js backend (Part A) and your frontend's communication with it (Part B).
    - **Include:**
        - Description of your testing approach (e.g., manual testing).

        - Specific test cases you used (e.g., different inputs, edge cases, error conditions).
        - Expected results and actual results of your tests.
        - Screenshots or logs demonstrating your testing process and results.
        - Any challenges you encountered and how you addressed them.
    - **Clarity:** Ensure your report is well-organized, easy to understand, and provides sufficient detail to demonstrate the thoroughness of your testing.

Submission Requirements:

- **`docker-compose.yml` (and `Dockerfile` if needed):** A `docker-compose.yml` file that defines how to build and run your application in a Docker container. If a separate `Dockerfile` is needed, include that as well. Your application should be fully functional when launched using `docker compose up`. The application should be accessible at `http://localhost:3000`
- **`server.js`:** Your Node.js server code.
- **`posting.html`:** Your HTML page with embedded JavaScript.
- **`report.pdf`:** Your test report.

Grading:

- **Part A (Node.js Backend):** 40 points
- **Part B (HTML Frontend):** 50 points
- **Part C (Test Report):** 10 points
- **Dockerization:** Failure to provide the required Docker files will result in 0 points for the entire assignment.
- **Code Functionality:** Correct implementation of the specified requirements.
- **Code Style and Readability:** Well-structured, commented, and easy-to-understand code.
- **Error Handling:** Proper handling of potential errors (e.g., file I/O errors, invalid input).
- **Asynchronous Operations:** Correct use of asynchronous JavaScript and `fetch` for client-server communication.
- **Real-time Updates:** Successful implementation of periodic updates to the displayed posts.

Important Notes:

- This assignment emphasizes the use of core Node.js, `express`, `body-parser`, vanilla JavaScript, and `fetch`. Avoid using any other external libraries or frameworks.
- Ensure your application is properly containerized using Docker, demonstrating your understanding of containerization principles.
- Thoroughly test your application to ensure it meets all requirements and handles various scenarios gracefully.