Assignment 4: Client-Server Posting System with MySQL2 Data Storage

Objective

You will build a basic client-server posting system using Node.js, jQuery, and jQuery UI. The system will:

- **Accept and store posts:** Each post must have a unique ID, a topic, data, and a timestamp. Posts must be saved persistently in a MySQL database using **mysql2**.
- **Accept and store responses:** Each response must have a unique ID, be associated with a post (via a postId), contain data, and include a timestamp. These should also be stored in MySQL using **mysql2**.
- **Serve data to a frontend:** The data will be fetched and updated asynchronously so that all users see new posts and responses. **Updates from any user must be visible to all users with a delay of less than 300 milliseconds.**
- **Undergo load testing:** The system will be tested for functionality and performance (load testing) using npm loadtest.

Technology Stack

- **Backend:** Node.js with MySQL integration using the **mysql2** module
- **Frontend:** HTML, jQuery, and jQuery UI
- **Communication:** AJAX calls (via jQuery) to your Node.js server
- **Containerization:** Docker (Dockerfile and/or docker-compose.yml)

Database Schema

Create a MySQL database that includes at least two tables:

- **posts:** Columns should include `id` (auto-increment primary key), `topic`, `data`, and `timestamp`.
- **responses:** Columns should include `id` (auto-increment primary key), `postId` (foreign key referencing `posts.id`), `data`, and `timestamp`.

Project Structure

Submit a single, compressed archive (.zip) containing the following files:

- `docker-compose.yml` (and Dockerfile if needed; ensure both the Node.js server and MySQL container are configured)
- `server.js`
- `posting.html`
- `package.json`
- `report.pdf`

Part A: Node.js Backend with MySQL2 Integration (40 Points)

Database Connection

- **Establish a connection** from your Node.js server to the MySQL database using the **mysql2** module.
- Ensure proper handling of connection errors and use environment variables for database credentials where possible.

Endpoints

1. **POST /postmessage**
    - **Input:** Accepts `topic` and `data` from the client.
    - **Process:** Inserts a new record into the `posts` table with an auto-generated unique ID and a timestamp.
    - **Output:** Returns a JSON response `{ success: true, id: newPostId }` where `newPostId` is the ID of the inserted post.
2. **POST /postresponse**
    - **Input:** Accepts `postId` and `data`.
    - **Process:** Inserts a new record into the `responses` table. Validate that the provided `postId` exists in the `posts` table.
    - **Output:** Returns a JSON response `{ success: true, id: newResponseId }` where `newResponseId` is the ID of the inserted response.
3. **Optional:** GET `/alldata`
    - **Purpose:** Retrieve all posts along with their corresponding responses.
    - **Implementation:** You may use JOIN queries or perform separate queries to fetch and combine the data.

Error Handling & Validation

- **Validate Input:** Check that all required fields (e.g., `topic`, `data`, and a valid `postId`) are provided.
- **Handle Errors:** Gracefully handle errors from MySQL queries or connection issues and return appropriate error messages to the client.

Performance Requirement

- **Timely Propagation:** Your backend should be designed so that when a post or response is created, it becomes available to any other connected client within **300 milliseconds**. Consider using efficient query design, proper indexing, and ensuring low-latency communication between the server and the MySQL database.

Allowed Modules

- `express`
- `body-parser`
- **mysql2**
- Standard Node.js built-in modules (e.g., `fs`, `Date`)
- **Note:** `npm loadtest` is allowed only for testing purposes; no other external libraries for functionality are permitted.

Part B: HTML Frontend (50 Points)

posting.html

- **AJAX & jQuery UI:** Use jQuery for AJAX requests and DOM manipulation, and incorporate at least one jQuery UI widget (e.g., dialog, accordion, datepicker) to enhance the user interface.
- **Display Data:** Fetch and display all posts and their responses (e.g., via GET `/alldata`). The UI should reflect new posts or responses from any user within **300 milliseconds** of creation.
- **Create New Posts:** Provide a form (or jQuery UI dialog) for submitting new posts:
    - Send a POST request to `/postmessage` with the `topic` and `data`.
    - Dynamically update the list of posts upon successful creation.
- **Create New Responses:** For each post, include an option to add a response:
    - Send a POST request to `/postresponse` with the `postId` and `data`.
    - Dynamically update the responses for that post upon successful submission.
- **Asynchronous Updates:** All updates must be applied without a full-page reload, and the changes (whether initiated by the current user or another user) should appear to all users with a latency of less than 300 milliseconds.

Part C: Test Report (10 Points)

Overall Testing Approach

- **Backend Testing:** Describe how you tested your Node.js endpoints, the MySQL2 integration, data structure, error handling, and the responsiveness of data updates.
- **Frontend Testing:** Explain how you verified the creation, display, and updating of posts/responses, with special emphasis on ensuring that updates are propagated within 300 milliseconds.

Functional Test Cases

- **Coverage:** Test with both normal and edge-case inputs (e.g., invalid data, missing fields, invalid `postId`).
- **Documentation:** Provide expected versus actual results, supported by screenshots or logs.

Load Testing with npm loadtest

- **Setup:** Install loadtest (e.g., using `npm install -g loadtest` or `npx loadtest`).
- **Execution:** Run load tests against your server.
- **Report:** In your `report.pdf`, include:
    - The command(s) used (e.g., `loadtest --concurrency=10 --requests=1000 http://localhost:3000/postmessage`).
    - Results (e.g., requests per second, response times, error details).
    - A brief analysis of server performance under load, including observations on the update latency.

Challenges and Solutions

- Document any issues encountered (e.g., with MySQL2 connectivity, query performance, low-latency updates, or Docker configuration) and how you resolved them.

Submission Requirements

- **docker-compose.yml (and Dockerfile if needed):** Must configure both the Node.js server and the MySQL container so that the application runs with `docker compose up`.
- **server.js:** Node.js server code implementing MySQL2 integration, API endpoints, and query handling.
- **posting.html:** Frontend code utilizing jQuery and jQuery UI for asynchronous updates and UI interactions.
- **package.json:** Must include all project dependencies and scripts required to run your application.
- **report.pdf:** A detailed test report including load testing commands, results, analysis, and documentation of functional test cases, with special attention to the under-300-millisecond update requirement.

Grading Matrix (Total 100 Points)

Part A: Node.js Backend with MySQL2 Integration	**(40 Points)**
1. POST /postmessage endpoint (inserting new posts)	10
2. POST /postresponse endpoint (inserting responses)	10
3. Database Schema & MySQL2 Integration (unique IDs, timestamps, relational integrity)	10
4. Error Handling & Validation (invalid postId, missing fields, etc.)	5
5. Docker Configuration for Backend and MySQL (e.g. docker-compose.yml)	5
Part B: HTML + jQuery/jQuery UI Frontend	**(50 Points)**
1. Displaying Existing Posts/Responses (fetched via AJAX, rendered in UI)	10
2. Creating New Posts (async form submission, auto-update UI)	10
3. Creating New Responses (async form submission per post, auto-update UI)	10
4. Use of jQuery UI Component(s) (e.g., dialog, accordion, etc.)	10
5. Overall User Experience, Code Clarity, and Update Latency (<300ms)	10
Part C: Test Report	**(10 Points)**
1. Test Approach & Functional Test Cases (screenshots/logs, thoroughness)	3
2. Load Testing with npm loadtest (commands, results, analysis)	3
3. Clarity & Presentation (challenges, solutions, well-structured report)	4
Total	**100**

Important Notes

- **Required Modules:** Use only the allowed modules on the server side (`express`, `body-parser`, **mysql2**, and Node.js built-ins).
- **Environment Configuration:** Ensure your MySQL database credentials and connection details are properly configured (preferably via environment variables).
- **Low Latency Requirement:** Your application must ensure that posts and responses created by any user are visible to all other users within 300 milliseconds.
- **Docker Setup:** Your Docker configuration must set up both the Node.js application and the MySQL database. The application should be accessible at `http://localhost:3000`.
- **Testing:** Thoroughly test your solution under various conditions, including high concurrency and load testing.
- **Failure to provide project.json and docker-compose.yml (and dockerfiles) will result in 0 points for the entire assignment.**