Lab Practice: Building a Simple Task Management Application with Node.js, Express.js, and SQLite

Description:

The Task Management Application is a simple web-based tool designed to help users organize and keep track of their tasks. It provides a basic interface to:

- Add Tasks: Users can input a description of a task and add it to their list.
- View Tasks: All added tasks are displayed in a list format.
- Mark Tasks as Complete: Users can mark tasks as finished, visually indicating their completion.
- Delete Tasks: Users can remove tasks from their list if they are no longer needed.

Objective:

- Set up a basic Node.js server using Express.js.
- Create routes to display tasks, add new tasks, mark tasks as complete, and delete tasks.
- Connect and query a SQLite database to store task information.

Part 1: Set Up the Environment

1. Create a project folder:

   ```
   mkdir task-manager

   cd task-manager
   ```

2. Initialize a new Node.js project

   ```
   npm init -y
   ```

3. Install dependencies:

   - ***Express.js*** for the server
   - ***sqlite3*** for database access

   ```
   npm install express sqlite3 body-parser
   ```

Part 2: Set Up the Database

4. Create a script to initialize the database:

   - Create a file named ***setup_db.js*** with the following code:

   ```
   const sqlite3 = require('sqlite3').verbose();
   const db = new sqlite3.Database('tasks.db');

   db.serialize(() => {
   ```

```
    db.run(`
      CREATE TABLE IF NOT EXISTS tasks (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        description TEXT NOT NULL,
        completed INTEGER DEFAULT 0
      )
    `);
  });
  db.close();
  console.log("Database setup complete.");
```

5. Run the setup script:

```
node setup_db.js
```

- Confirm that a ***tasks.db*** file is created in your project folder.

Part 3: Create the Web Server

6. Create ***app.js*** in your project folder:

```
const express = require('express');
const sqlite3 = require('sqlite3').verbose();
const bodyParser = require('body-parser');

const app = express();
const PORT = 3000;

app.use(bodyParser.urlencoded({ extended: true }));
app.use(express.static('public')); // For serving static files (e.g., CSS)

// Database connection function
function getDB() {
  return new sqlite3.Database('tasks.db');
}

// Route: Display all tasks
app.get('/', (req, res) => {
  const db = getDB();
  db.all('SELECT * FROM tasks', [], (err, rows) => {
    if (err) {
      res.status(500).send("Error retrieving tasks");
      return;
    }
    let html = `
      <!DOCTYPE html>
      <html>
      <head>
        <title>Task Manager</title>
        <link rel="stylesheet" href="style.css">
      </head>
      <body>
        <h1>Task Manager</h1>
        <form action="/add" method="post">
          <input type="text" name="description" placeholder="Add a new task" required>
          <button type="submit">Add Task</button>
        </form>
        <ul>
    `;

    rows.forEach(task => {
      html += `
          <li>
            <form action="/complete/${task.id}" method="post" style="display: inline;">
              <button type="submit" ${task.completed ? 'disabled' : ''}>${task.completed ?
'Completed' : 'Mark Complete'}</button>
            </form>
            ${task.description}
            <form action="/delete/${task.id}" method="post" style="display: inline;">
              <button type="submit">Delete</button>
            </form>
          </li>
        `;
    });

    html += `
        </ul>
      </body>
      </html>
    `;

    res.send(html);
    db.close();
  });
});
```

```
// Route: Add a new task
app.post('/add', (req, res) => {
  const description = req.body.description;
  const db = getDB();

  db.run('INSERT INTO tasks (description) VALUES (?)', [description], (err) => {
    if (err) {
      res.status(500).send("Error adding task");
      return;
    }

    res.redirect('/');
    db.close();
  });
});

// Route: Mark a task as complete
app.post('/complete/:id', (req, res) => {
  const taskId = req.params.id;
  const db = getDB();

  db.run('UPDATE tasks SET completed = 1 WHERE id = ?', [taskId], (err) => {
    if (err) {
      res.status(500).send("Error marking task as complete");
      return;
    }

    res.redirect('/');
    db.close();
  });
});

// Route: Delete a task
app.post('/delete/:id', (req, res) => {
  const taskId = req.params.id;
  const db = getDB();

  db.run('DELETE FROM tasks WHERE id = ?', [taskId], (err) => {
    if (err) {
      res.status(500).send("Error deleting task");
      return;
    }
    res.redirect('/');
    db.close();
  });
});

app.listen(PORT, () => {
  console.log(`Server running at http://localhost:${PORT}`);
});
```

Part 4: Create the Style Sheet (public/style.css)

7. Create a folder named ***public*** in your project directory.
8. Create a file named ***style.css*** inside the public folder.

```
body {
  font-family: sans-serif;
  margin: 20px;
}

input[type="text"] {
  padding: 8px;
  margin-right: 10px;
  border: 1px solid #ccc;
}

button {
  padding: 8px 12px;
  background-color: #4CAF50;
  color: white;
  border: none;
  cursor: pointer;
}

button:disabled {
  background-color: #ccc;
  cursor: not-allowed;
}

li {
  list-style-type: none;
  padding: 8px;
  border-bottom: 1px solid #eee;
}
```

Running the Server and Testing

9. Start the server:

```
node app.js
```

10. Open your web browser and visit ***http://localhost:3000/.***

   You should see the Task Manager application with an input field to add new tasks and a list of existing tasks (if any).

Exercises and Extensions:

- **Add Due Dates:** Modify the database schema and application to include due dates for each task.
- **Task Prioritization:** Add a priority field to tasks (e.g., High, Medium, Low) and display tasks sorted by priority.
- **Error Handling:** Implement more robust error handling and display user-friendly error messages.
- **CSS Styling:** Improve the visual appearance of the application with more advanced CSS.
- **Client-Side JavaScript:** Use JavaScript to handle task completion and deletion on the client-side without full page reloads.