|  |  |
| --- | --- |
| **Code Review Checklist** | |
| **Author:** | Mark Zuckerberg |
| **Reviewer:** | Gaurav Patel |
| **Date Reviewed:** | 2024-11-03 |
|  | |
| **Code Style Guidelines** | |
| **Indentation** | * The die statement should be indented by 2 spaces from catch. * The statements directly after each if, else if, and else statement ($dbh->query, $command, $stmt, $success, etc.) should all be consistently indented by 2 spaces. * For the HTML output inside PHP (echo "<tr><td>{$students[$i]['id']}</td>"...), each <td> should ideally align consistently (e.g., breaking lines for readability and indenting 2 spaces within each <tr>). * Each line inside the <script> tag should be indented by 2 spaces for consistency with your overall code style. |
| **Code Blocks** | * In the try and catch block: The opening curly brace { is on the same line as try. |
| **Snake Case** | * $selectCommand and $selectStmt should be renamed to $select\_command and $select\_stmt for snake\_case consistency. * $editStudent should be renamed to $edit\_student. * refreshCSS could be renamed to refresh\_css. |
| **Comments** | * There is no comment explaining that the try block is attempting to establish a connection to the MySQL database using PDO, and the catch block handles any exceptions if the connection fails. * Each conditional block that handles different actions (delete, add, edit, commit\_edit, and the default case) lacks comments explaining what each block is intended to do. * After executing the query to select all students, there is no comment explaining that the following loop fetches each student record and stores it in the $students array. * There is no comment before the HTML table indicating that it displays the list of students retrieved from the database. Within the HTML table, the for loop that iterates over $students lacks a comment explaining that it generates table rows for each student. * The script injected by live-server at the end of the HTML lacks a comment explaining its purpose, which is to enable live reloading during development. |
| **Code Structure Guidelines** | |
| **Modularization** | * Move CSS and JavaScript code to separate files to improve readability and maintainability. Link the external CSS file in the HTML <head> section to keep styling separate, and link the JavaScript file at the end of the HTML document for better loading performance. This approach keeps HTML focused on content, CSS on styling, and JavaScript on behavior. * Place the form handling logic for actions like adding, editing, and deleting students in dedicated PHP files. Set the form's action attribute to target these files, allowing each file to handle a specific action. This separation keeps the main file streamlined, improves organization, and reduces the risk of errors by isolating each action in its respective file. * Move all database operations (such as retrieving, adding, deleting, and editing students) into a separate file with individual functions for each action. This modular approach centralizes database logic, making it reusable and easier to update or debug without modifying the main application structure. Each function can be included in other parts of the codebase as needed, enhancing maintainability and scalability. |
| **SQL Injection** | * The DELETE statement in if ($act == "delete") is directly using id in the query. Since $id is concatenated directly into the SQL statement without parameterization, this line is vulnerable to SQL injection. |
| **Input Filtering** | * $gpa: Directly accessed from $\_POST["gpa"] without sanitization or validation. It should be validated as a numeric value and possibly checked to be within a valid GPA range * $\_POST Direct Access: $gpa is accessed directly from $\_POST, unlike other inputs that use filter\_input. For consistency and security, avoid direct access by using filter\_input for all input values. * $act : While $act is sanitized, it should also be restricted to only expected values (delete, add, edit, commit\_edit) to prevent unintended actions. |
| **Separation Of Concerns** | * PHP should not output HTML elements as part of a string. Does this occur anywhere? * In both the edit and add sections, PHP is directly embedding HTML elements (like <label> and <input>) within echo statements, which also mixes PHP and HTML output. * Conditional statements (such as checking $act == "edit") use PHP to dynamically output HTML form structures directly, resulting in HTML being embedded as part of PHP logic. |
| **Code Duplication** | * The HTML <table> structure inside PHP (<tr><td>{$students[$i]['id']}</td>...) repeats for each student. Instead of echoing HTML in PHP, consider generating the HTML outside the PHP loop and using a function or reusable block for each row. * The block checking if ($success) and iterating over $stmt->fetch() is repeated. Consider wrapping this logic in a function, which can be called whenever you need to process query results * The form structure for adding and editing students has several repeated elements (<label>, <input>, etc.). You could create a function to generate the form, passing values as parameters, and use it for both the “add” and “edit” cases to avoid duplication. |
| **Functionality** | |
| **Requirements Met** | * When trying to edit a student, if an invalid or non-existent id is provided, $editStudent might be false or null since fetch() can return false when no record is found. Accessing $editStudent['first\_name'], $editStudent['last\_name'], or $editStudent['GPA'] in the form will result in an error. Consider adding a check to ensure the student exists before attempting to edit. * In the edit form, the field for GPA uses <?php echo $editStudent['GPA'] ?>, but the correct key should be gpa (lowercase) based on the database structure. This mismatch will cause an "undefined index" error, resulting in the GPA not displaying correctly. |