User Stories and Code Review

User Stories:

1. As a user, I want to see a list of available genres on the website's home page, so that I can easily navigate to games of my preferred genre.

2. As a user, I want to view a specific genre's page, so that I can explore games belonging to that genre and learn more about them.

3. As a user, I want to see details about a game when I select it from a genre's page, including its title, header, and content.

4. As a user, I want to be able to switch between different visual styles (dark mode, light mode, Neumont mode) for the website to customize my browsing experience.

5. As a user, I want the ability to edit and update the title, header, and content of a page for a specific game, so that I can keep the information accurate and up-to-date.

6. As a user, I want to be able to add new pages with titles, headers, and content, so that I can contribute new information about games to the website.

7. As a user, I want the option to delete or restore pages, allowing me to control which content is visible on the website.

Code Review:

Overall, the code appears to be functional and structured. Here are some observations and recommendations:

1. Database Connection and Security:

- It's good to define constants for database credentials but consider using environment variables or a configuration file instead to avoid hardcoding sensitive information.

- Using prepared statements can help prevent SQL injection attacks. Consider implementing them for database queries.

2. Code Organization:

- The code snippets are relatively well organized into different files (e.g., `dbConnector.php`, `Helper.php`, `EditPage.php`, etc.), which is a good practice for maintainability.

3. Function Naming and Comments:

- Function names like `GetChildPages`, `GetAllPages`, etc. are descriptive, but consider adding comments or docstrings to explain what each function does.

- Adding comments to complex sections of code or explaining the purpose of certain steps can improve code readability.

4. Error Handling:

- In the `GetConnection` function, using `die()` to handle database connection errors is a simple approach, but you might consider implementing more robust error handling mechanisms, such as exceptions.

5. User Interface:

- The UI is functional but lacks styling. Consider enhancing the visual design of the website to make it more appealing and user-friendly.

6. HTML Form Usage:

- In the forms, consider providing labels for input fields (`<label for="Title">Title:</label>`) for better accessibility and user experience.

7. SQL Query Building:

- When constructing SQL queries, use parameter binding for dynamic values to prevent SQL injection vulnerabilities.

8. Data Validation:

- Ensure that user inputs are properly validated and sanitized before using them in database queries to avoid potential security issues.

9. Code Reusability:

- There are instances where certain code sections appear to be duplicated. Consider refactoring and extracting common functionality into reusable functions.

10. Cookie Handling:

- Make sure to properly validate and sanitize user input, such as the style preference, before using it to set cookies.

11. Page Navigation:

- Consider adding a navigation structure to help users easily navigate between different pages and sections of the website.