To create a modular, maintainable, and well-documented website, you can follow these best practices:

1. \*\*Separate concerns\*\*: Divide your code into logical units, where each unit is responsible for a single aspect of the website. For example, separate HTML, CSS, JavaScript, and PHP files, and group related functionality in separate files and folders.

2. \*\*Follow a consistent folder structure\*\*: Organize your files and folders in a consistent and logical manner. A common structure is:

- `assets`: for images, fonts, and other static resources.

- `css`: for all CSS files.

- `js`: for JavaScript files.

- `php`: for server-side PHP files (you can further categorize them into `includes`, `controllers`, and `models`).

- `templates`: for HTML templates or partials.

- `sql`: for SQL scripts and database schema files.

3. \*\*Use a version control system\*\*: Use a version control system like Git to track changes, collaborate with other developers, and maintain a history of your project.

4. \*\*Use a build system or task runner\*\*: Tools like Gulp, Grunt, or Webpack can help automate tasks like minifying and concatenating JavaScript and CSS files, optimizing images, and compiling templates. This will improve your site's performance and maintainability.

5. \*\*Leverage PHP frameworks\*\*: Using a PHP framework like Laravel or Symfony can help you create a well-structured and maintainable application. These frameworks provide a solid foundation and follow best practices, which can be helpful when developing complex applications.

6. \*\*Use front-end frameworks and libraries\*\*: Popular front-end libraries and frameworks like React, Angular, or Vue.js can help you build modular and reusable UI components, making it easier to maintain and scale your front-end code.

7. \*\*Document your code\*\*: Write clear and concise comments to explain the purpose of your code, especially when it may not be obvious. Document any assumptions, dependencies, and potential issues. Consider using tools like JSDoc or phpDocumentor to generate API documentation from your code comments.

8. \*\*Write modular and reusable code\*\*: Encapsulate functionality in small, self-contained functions or classes. This will make your code easier to understand, maintain, and test.

9. \*\*Adopt a coding style guide\*\*: Use a consistent coding style across all your files, and enforce it using tools like ESLint or PHP CodeSniffer. This will make your code easier to read and maintain.

10. \*\*Test your code\*\*: Write unit tests and integration tests for your code to ensure it works as expected and to catch potential issues early. Using a continuous integration (CI) tool can help automate the testing process.

By following these best practices, you'll create a modular, maintainable, and well-documented website that is easy to work with and scale.