Brocamp Project Implementation Guidelines to follow

Consistent Naming Conventions:

- Use meaningful and descriptive names for variables, functions, class names and database entities that reflect their purpose. (e.g., create_user, update_post, delete_comment)
- Follow a consistent naming convention such as camelCase or snake_case throughout your codebase.

Consistent Error Messaging:

- Provide clear and concise error messages to users when validation errors occur.
- Use descriptive error messages that explain what went wrong and how to correct the issue (e.g., "Please enter a valid email address").
- Display error messages near the corresponding form fields or in a prominent location on the page for easy visibility.

Confirmation on deleting data: Consider adding confirmation steps before deleting data to prevent accidental loss.

Secure Authentication Mechanisms:

- Use secure authentication mechanisms such as Bcrypt for password hashing and salting to protect user passwords from being compromised.
- Avoid storing passwords in plaintext or using weak hashing algorithms (e.g., MD5, SHA-1) that are susceptible to brute force attacks.

Error Handling:

- Instead of showing messages like “Error”, Provide informative error messages to users when login attempts fail due to incorrect credentials, account lockout, or other reasons. (Like Invalid username or password, Username already exists etc).
- Avoid revealing sensitive information in error messages that could aid attackers in exploiting vulnerabilities (e.g., "Invalid username" instead of "Invalid username or password").

Password Strength Requirements:

- Implement password strength validation to enforce strong passwords that include a mix of uppercase and lowercase letters, numbers, and special characters.
- Provide clear guidelines on password requirements (e.g., minimum length, character types) to help users create secure passwords.

On File Uploads, Limit File Types and Sizes:

- Restrict the types of files that users can upload to prevent the upload of potentially malicious file types (e.g., executables, scripts).
- Set limits on the maximum file size to prevent denial-of-service attacks and ensure that your server can handle the uploaded files efficiently.

On File Uploads, Validate File Content:

Use file signature (magic number) detection or MIME type validation to verify that uploaded files match their declared file types.

Secure File Storage:

- Store uploaded files outside of the web root directory to prevent direct access to uploaded files by unauthorized users.
- Use unique file names or generate random file identifiers to prevent filename guessing attacks and ensure file uniqueness.
- Implement access control mechanisms to restrict access to uploaded files based on user permissions and roles.

On File uploads, Provide Feedback to Users:

- Provide clear and informative feedback to users during the file upload process, including progress indicators, success messages, and error messages.
- Use client-side validation to validate file size and type before initiating the upload process to avoid unnecessary server requests and improve user experience.

Optimize File Upload Performance:

Use asynchronous file upload techniques (e.g., AJAX, Fetch API) to upload files in the background without blocking the main thread and improve perceived performance.

Secure File Transmission:

- Use secure protocols such as HTTPS to encrypt file uploads and protect against eavesdropping and tampering during transmission.
- Implement server-side validation to prevent CSRF (Cross-Site Request Forgery) attacks by validating the authenticity of file upload requests.

On File uploads, Sanitize File Names:

Sanitize file names to remove potentially dangerous characters or sequences that could be used to exploit file system vulnerabilities (e.g., directory traversal attacks).

UI Standards

- **Consistency**: Maintain a consistent look and feel across all pages in your web app. This includes using the same color scheme, fonts, button styles, and layout patterns. Consistency helps users learn the application quickly and navigate efficiently.
- Avoid inactive or unnecessary links throughout the project.
- Design your web app to be responsive and adaptive, ensuring a consistent user experience across different devices and screen sizes.
- Use flexible layouts, fluid grids, and media queries to adjust content and layout based on viewport dimensions and breakpoints.
- Provide clear and intuitive navigation menus and controls to help users navigate through your web app easily.
- Set cursor style to pointer for all links without hrefs
- Regarding CSS, always use external stylesheets, not internal or inline.

On listing data on Admin Panels, Use Pagination:

- Implement pagination to limit the number of entries displayed per page, improving performance and user experience, especially for large datasets.
- Provide navigation controls (e.g., pagination links, "Previous" and "Next" buttons) to allow users to navigate between pages of results.

On listing data on Admin Panels, Use Search and Filtering:

- Include search functionality to allow users to search for specific entries based on keywords or criteria.
- Implement filtering options to enable users to refine the list of entries based on specific attributes or categories.

On listing data on Admin Panels, Implement Sorting Feature:

- Allow users to sort the list of entries based on different attributes (e.g., alphabetical order, date created, relevance) in ascending or descending order.
- By default, listing of data should be in descending order.
- Provide intuitive UI controls (e.g., dropdown menus, clickable column headers) for users to select and apply sorting criteria.

GIT Standards to follow

General Workflow:

- **Feature Branches:** Use feature branches for development. This isolates changes for specific features and facilitates code reviews before merging into the main codebase (often called master or main).

- **Commit Messages:** Write clear and concise commit messages that describe the changes made in each commit. Use a consistent format like "feat: Added new login feature" or "fix: Resolved bug in user profile section." Follow a consistent format (e.g., imperative mood, 50-character subject line, optional body with additional details) to make commit messages informative and easy to understand.

- **Push Regularly:** Push your code changes to the remote repository regularly to avoid losing work and facilitate collaboration.

- **Use .gitignore file:** If you want to make some files not tracked by git, you have to create and add the filename in the .gitignore file. Never commit unnecessary files, use .gitignore file for making those files not tracked by git.