Learning Hub Website

Static Code Review Report

[LH_SCR_REPORT]

✧ **Project Name**	Learning Hub Website
✧ **Files Reviewed**	LH_CODE
✧ **Review Sheet**	The review sheet can be accessed by clicking here.
✧ **Review Scope**	Static review for the code, excluding unit testing and security aspects.

Executive Summary:

- Overall code pass rate : **91%**
- Modules below 90%: **None**
- Area of Improvement: **Code structure and Validation/ UX**
- Recommendation actions: **Function refactoring, validation checks, replace hardcoded values**

Introduction:

This report presents the results of a static code review conducted across various modules of the system. The review process was based on a standardized checklist assessing adherence to code quality, maintainability, and alignment with the project's architecture and functional requirements.

Each feature/module was reviewed independently, with checklist items evaluated and marked as either passed or failed. The goal is to ensure consistency, readability, and reliability across the code-base before integration or deployment.

This report includes a dashboard summarizing overall code quality status, highlights areas that require improvement, and provides actionable recommendations.

Code Quality Threshold:

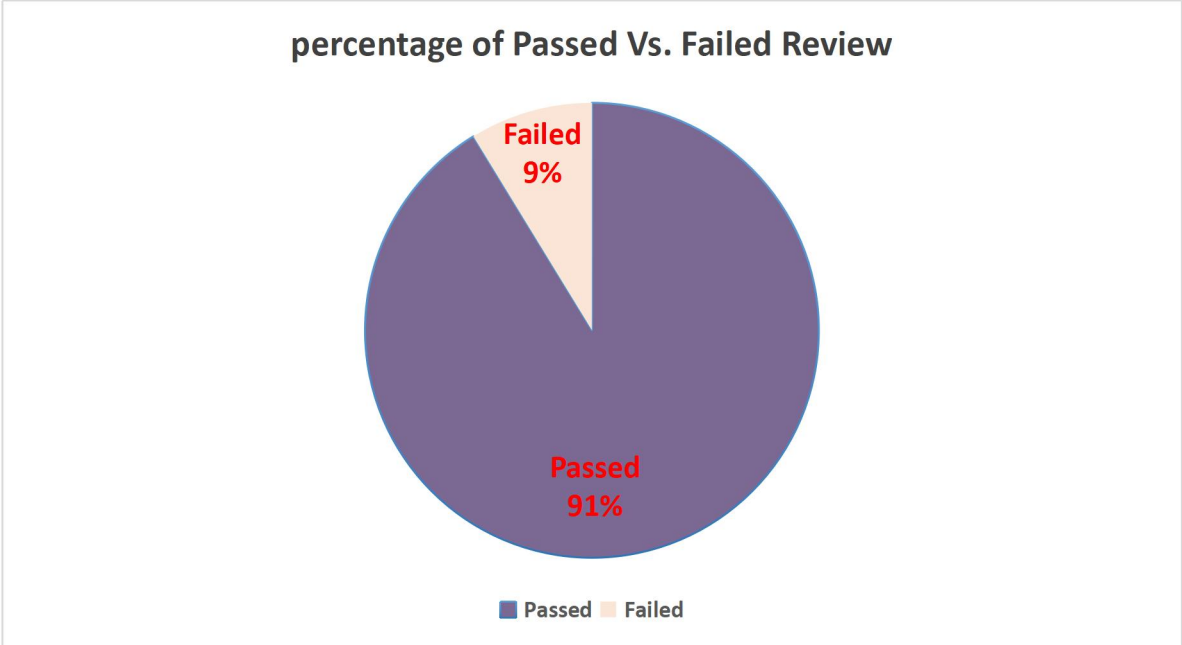
To maintain acceptable code quality standards, a minimum pass rate of 80% (i.e., a maximum failure rate of 20%) is required per module and a minimum pass rate of 90% overall.

Modules falling below this threshold are flagged for review and must be re-factored before proceeding to the next stage in the development cycle.

Summary Table (Per Module):

Module name	Total items	Passed	Failed	Pass %	Status
Registration	16	14	2	87%	Pass
Login	16	15	1	93%	Pass
UserHome	16	16	0	100%	Pass
Categories	16	14	2	87%	Pass
PublishArticle	16	13	3	81%	Pass
PublishAudio	16	14	2	87%	Pass
PublishVideo	16	14	2	87%	Pass
AdminHome	16	16	0	100%	Pass
DeleteUser	16	16	0	100%	Pass
DeletePosts	16	14	2	87%	Pass
Total	160	146	14	91%	Pass

Dashboard for Passed VS. Failed



Category Breakdown and Area of Development

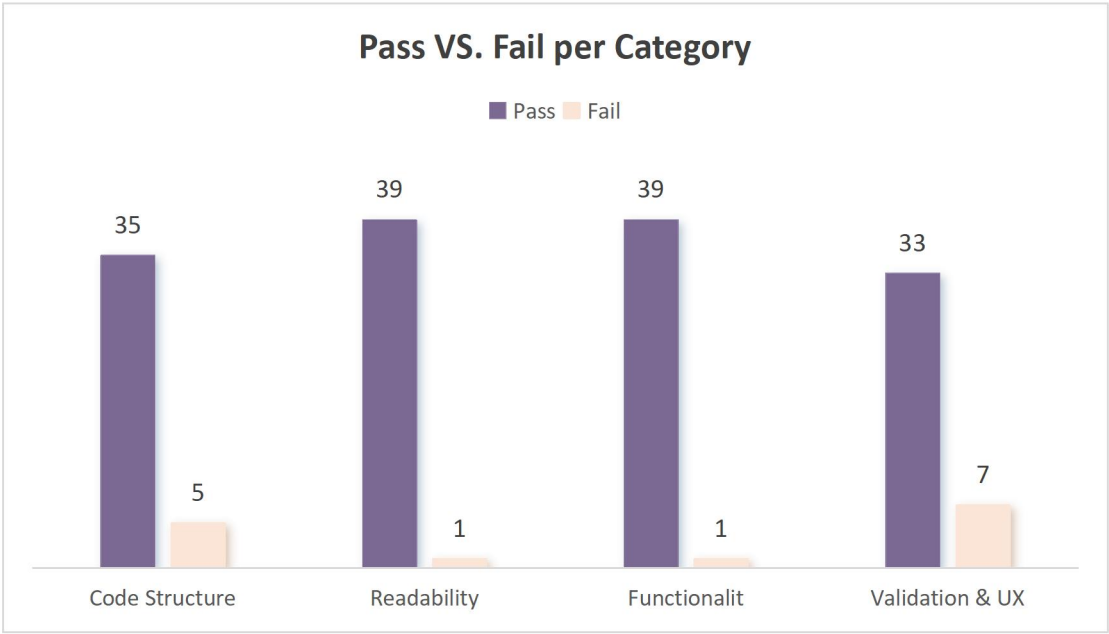
To provide a more focused analysis, checklist items were grouped into four key review categories. Each category targets a specific aspect of the code to ensure quality, consistency, and maintainability.

Category	Focus Area	Checklist Items Covered
Code Structure	Project organization, folder conventions, reuse of code	- Project architecture follows standards - Folder/file naming conventions - Reusable functions are used
Readability	Naming, formatting, and code clarity	- Clear variable and function names - Proper formatting and indentation - Necessary and clear comments
Functionality	Logical correctness, alignment with requirements, expected UI behaviors	- Code meets functional requirements - Logic is complete and correct - UI behaviors are correct
Validation & UX	Input validation, user experience, navigation logic	- Input values are validated - Navigation and buttons work as expected - No hardcoded values

Category Review Summary

The table below shows how many checklist items passed or failed in each category based on the static code review across all modules.

Category	No of Modules	No of items per category	Pass	Fail	% Pass
Code Structure	10	4	35	5	87.5%
Readability			39	1	97.5%
Functionality			39	1	97.5%
Validation & UX			33	7	82.5%



Conclusion

The static code review revealed that the most significant areas for improvement are Code Structure and Validation/UX. Specifically:

- ✧ **In Code Structure**: many functions are too long or perform multiple tasks, violating the Single Responsibility Principle (SRP).
- ✧ **In Validation/UX:** common issues include hard-coded values, unused code, and missing basic input validation.

These findings suggest a need for more consistent adherence to modular design and user input handling standards across modules.

Recommendation for Future Releases

- ✧ **Code Structure**

- ✓ Refactor lengthy or multitasking functions into smaller, focused ones (SRP).
- ✓ Use modular design by grouping related functionality.
- ✓ Ensure code is easy to maintain and extend by following the defined architectural structure.

- ✧ **Validation/ UX**

- ✓ Remove unused variables and redundant code to reduce clutter.
- ✓ Validate user inputs properly—ensure required fields are checked and value constraints are enforced.
- ✓ Replace hardcoded values with centralized constants or configuration files.
- ✓ Ensure UI actions (like form submissions or navigation) behave reliably and consistently.