Code Review Checklist

This checklist verifies code quality across formatting, comments, error handling, and more. They're critical for building secure, resilient software systems.

STEP 1

Formatting

- [ ] Consistent styling guide

  Ensure that the code follows the established styling guide, including consistent indentation, spacing, and naming conventions.

  Use automated formatters to catch style violations early and maintain code consistency.

- [ ] Code clarity

  Review the code's formatting to ensure clarity and readability, making the logic flow easier to follow.

  Enforce style rules through linter integrations in IDEs and CI/CD pipelines for consistent formatting across contributors.

STEP 2

Comments

- [ ] Clear Explanation

  Include clear comments that explain the intent and approach of complex code sections.

  Ensure comments clarify the reasoning behind design decisions, aiding in knowledge transfer among team members.

- [ ] Reduced Dependency

  Use comments to reduce dependency on individual developers by documenting critical information.

  Ensure that comments are comprehensive and ensure that critical information is accessible to the entire team.

STEP 3

Error Handling

- [ ] **Robust Handling**
  Implement robust error handling to provide graceful failures and meaningful error messages.

  Verify the correct catching and handling of exceptions and log errors with useful context.

- [ ] **Prevent Unhandled Exceptions**
  Ensure that error handling practices are consistent throughout the codebase.

  Proper exception handling and logging reduce crashes by gracefully dealing with edge cases and provide clear debugging information.

STEP 4

Security

- [ ] **Adherence to Security Best Practices**
  Review the code for adherence to security best practices such as input validation, output encoding/escaping, and encryption for sensitive data.

  Methodically inspect the code to identify areas where user input is not properly sanitised and where output needs correct encoding.

- [ ] **Protection Against Vulnerabilities**
  Check for vulnerabilities like SQL injection, cross-site scripting, and exposure of confidential data.

  Ensure that encryption is correctly applied using approved algorithms and key lengths for sensitive data, such as passwords or financial information.

STEP 5

Performance

- [ ] Efficiency Evaluation

  Evaluate algorithms and data structures for efficiency and look for opportunities to optimise unnecessary operations.

  Identify areas where memoization or caching can improve performance, particularly for repetitive computations.

- [ ] Optimisation Potential

  Be aware that optimisations, such as caching and algorithm improvements, can significantly speed up code, even by over 100 times in some cases.

  Keep in mind that even small improvements, such as a 10% boost in performance, can have a substantial impact when serving millions of requests.