Checklist

General

1. Am I able to understand the code easily?
Svar:

2. Is the same code duplicated more than twice?
Svar:

3. Can I unit test / debug the code easily to find the root cause?
Svar:

4. Is this function or class too big? If yes, is the function or class having too many responsibilities?
Svar:

Code formatting

5. Use alignments (left margin), proper white space. Also ensure that code block starting point and ending point are easily identifiable.
Svar:

6. Ensure that proper naming conventions (Pascal, CamelCase etc.) have been followed.
Svar:

7. Remove the commented code as this is always a blocker, while going through the code. Commented code can be obtained from Source Control (like SVN), if required.
Svar:

Architecture

8. Separation of Concerns followed:
- Split into multiple layers and tiers as per requirements (Presentation, Business and Data layers).
- Split into respective files (HTML, JavaScript and CSS).

Svar:

9. Design patterns: Use appropriate design patterns (if it helps), after completely understanding the problem and context.
Svar:

Coding best practices

10. No hard coding, use constants/configuration values.
Svar:

11. Group similar values under an enumeration (enum)
Svar:

12. Comments – Do not write comments for what you are doing, instead write comments on why you are doing. Specify about any hacks, workaround and temporary fixes. Additionally, mention pending tasks in your to-do comments, which can be tracked easily.
Svar:

13. Avoid multiple if/else blocks.
Svar:

Non Functional requirements

14. Readability: Code should be self-explanatory. Get a feel of story reading, while going through the code. Use appropriate names for variables, functions and classes. If you are taking more time to understand the code, then either code needs refactoring or at least comments have to be written to make it clear.
Svar:

15. Testability: The code should be easy to test. Refactor into a separate function (if required). Use interfaces while talking to other layers, as interfaces can be mocked easily. Try to avoid static functions, singleton classes as these are not easily testable by mocks.
Svar:

Object-Oriented Analysis and Design (OOAD) Principles

16. Follow Single Responsibility Principle (SRS): Do not place more than one responsibility into a single class or function, refactor into separate classes and functions.
Svar:

17. Follow Open Closed Principle: While adding new functionality, existing code should not be modified. New functionality should be written in new classes and functions.
Svar:

18. Follow Liskov substitutability principle: The child class should not change the behavior (meaning) of the parent class. The child class can be used as a substitute for a base class.
Svar:

19. Follow Interface segregation: Do not create lengthy interfaces, instead split them into smaller interfaces based on the functionality. The interface should not contain any dependencies (parameters), which are not required for the expected functionality.
Svar:

20 . Follow Dependency Injection: Do not hardcode the dependencies, instead inject them.
Svar: