Clean code guide

Code is clean if it can be understood easily – by everyone on the team. Clean code can be read and enhanced by a developer other than its original author. With understandability comes readability, changeability, extensibility, and maintainability.

"Any fool can write code that a computer can understand. Good programmers write code that humans can understand."

– Martin Fowler

General rules

- Don't be a Cowboy coder
- Follow standard conventions/best practices/style guides of any programming language and team/project related.
- Keep it simple stupid. Simpler is always better. Reduce complexity as much as possible.
- Boy scout rule. Leave the campground cleaner than you found it.
- Always find the root cause. Always look for the root cause of a problem.
- Do not DRY
- Do not override safeties
- Follow the Principle of Least Astonishment(POLA)
- Understand design patterns

Design rules

- Keep configurable data at high levels.
- Prefer polymorphism to if/else or switch/case.
- Separate multi-threading code.
- Prevent over-configurability.
- Use dependency injection.
- Follow the Law of Demeter. A class should know only its direct dependencies.

Understandability tips

- Be consistent. If you do something a certain way, do all similar things in the same way. Follow the **one word for each concept** rule. Do not use *fetch*, *retrieve*, and *get* for the same operation in different classes. Choose one of them and use it all over the project so people who maintain the codebase or the clients of your API can easily find the methods they are looking for.
- Use explanatory variables.
- Encapsulate boundary conditions. Boundary conditions are hard to keep track of. Put the processing for them in one place. Don't let them leak all over the code.
- Avoid logical dependency. Don't write methods which works correctly depending on something else in the same class.
- Avoid negative conditionals.

Names rules

Name should tell what the piece of code is doing and must be understandable on future enhancement. Details reference in the link

- Variable Naming -
  - Use descriptive and intention-revealing variable names
  - Make meaningful distinctions
  - Use pronounceable and searchable names
  - Avoid disinformation and encoded names
  - Avoid member prefixes or types information (Hungarian Notation)
  - Avoid mental mapping
  - Avoid generic name ie. temp
  - Replace Magic Numbers with Constants
- Route Naming
  - API endpoint name should be noun, kebab-case. More reference in the link
  - Use proper HTTP methods. Use Get, Put, Post, Patch, Delete properly.
  - Never use CRUD function names in URIs
- Function Naming
  - Function name should be verb, camelCase.
  - Do not use too long or too short name
  - For more link
- FIle and Folder naming
  - Controller name should be plural, camelCase
  - Model name should be singular, PascalCase
  - Project should be inside src folder
  - Follow project/team defined structure and naming conventions
- CSS naming
  - CSS class name and id should be camelCase

Functions rules (Reference Link)

- Small.
- Do one thing.
- Use descriptive names.
- Prefer fewer arguments.
- Have no side effects.
- Don't use flag arguments. Split method into several independent methods that can be called from the client without the flag.

Comments rules (Reference link)

- Always try to explain yourself in code.
- Don't be redundant.
- Don't add obvious noise.
- Don't use closing brace comments.
- Don't comment out code. Just remove.
- Use as explanation of intent.
- Use as clarification of code.
- Use as warning of consequences.

Source code structure

- Separate concepts vertically.
- Related code should appear vertically dense.
- Declare variables close to their usage.
- Dependent functions should be close.
- Similar functions should be close.
- Place functions in the downward direction.
- Keep lines short.
- Don't use horizontal alignment.
- Use white space to associate related things and disassociate weakly related.
- Don't break indentation.

Objects and data structures

- Hide internal structure.
- Prefer data structures.
- Avoid hybrids structures (half object and half data).
- Should be small.
- Do one thing.
- Small number of instance variables.
- Base class should know nothing about their derivatives.
- Better to have many functions than to pass some code into a function to select a behavior.
- Prefer non-static methods to static methods.

Tests

- One assert per test.
- Readable.
- Fast.
- Independent.
- Repeatable.

Code smells

- Rigidity. The software is difficult to change. A small change causes a cascade of subsequent changes.
- Fragility. The software breaks in many places due to a single change.
- Immobility. You cannot reuse parts of the code in other projects because of involved risks and high effort.
- Needless Complexity.
- Needless Repetition.
- Opacity. The code is hard to understand.

Final words. You can find lot of resources online related to clean code practice. If you want to start easily, here is one reference you can start today: ***Airbnb JavaScript Style Guide***