# Title: Frontend Pull Request Review Guidelines

**Follow Frontend best practices**

Ensure the best practices mentioned in [Front End Best Practices](https://confluence.freshworks.com/pages/viewpage.action?pageId=206574413) are followed. Any exceptions made to the practices would have to undergo thorough thought process and would be better to get opinion from senior frontend engineers

**Code Consistency**

Ensure consistency is followed everywhere through the code. Consistency in a code base makes code easier to read and understand, helps prevent bugs, and facilitates collaboration between regular and new developers.

**Legibility**

Ensure the code written is legible and easy to understand. Legible code is more reusable, bug-free, and future-proof. Look for proper and meaningful naming conventions for variable names, functions, class names etc...

**Accidental Errors/ Typos**

Accidental errors (e.g., typos) as well as structural errors (e.g., dead code, logic or algorithm bugs, performance or architecture concerns) are often much easier to spot and should be marked as comments for better readable code despite it having any impact on the way the code works.

**Security Leaks**

Look for security leaks such as Cross Site Scripting (XSS), CSRF etc...

Reference:

<https://www.smashingmagazine.com/2010/10/common-security-mistakes-in-web-applications/>

**Todos**

When the project is on aggressive timelines, sometimes the developer may be forced to write temporary business logic as a workaround. Ensure Todo comments tagging the developer is added to clearly state that it is only a workaround and code will be refactored in near future.

**Test cases**

No pull request should be approved without test cases unless and until there is absolutely no way to write test for the same. Any new component or entity introduced by the developer is expected to meet 100% code coverage and any code changes made to the existing code components should be covered by adding test cases to the existing test suite (if one exist and if not, the developer is responsible for setting up a new test suite for that existing component or any entity for that matter).

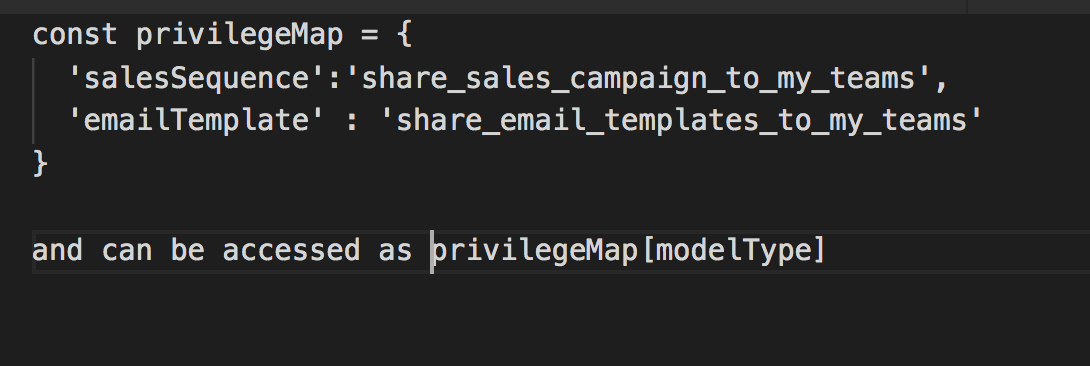
**Configurable Code**

Code that is configurable is always extensible and easier for new developers to make changes.

Example:



The multiple if else conditions in the above code snippet can be easily replaced with a map constant defined as



**Trading performance for one-liners is a big NO**

Performance should always be the top priority and should never be compromised for better looking one-liners.

Example;

      arrayVariable.map(...).filter(....).every(...) makes three iterations, while in fact it can be accomplished in a single loop.

**Tone of Review comments**

Avoid harsh review comments. Be courteous and respectful while also being very clear and helpful to the developer whose code you are reviewing.

Bad: "Why the hell did you hardcode z-index in your css ? "  
Good: "Using a z-index variable from variable.scss would be easier to maintain, when there's a change, instead of hardcoding."

**Encourage implementation reviews**

Code reviews often just focus on mistakes, but they should offer encouragement and appreciation for good practices, as well. It’s sometimes even more valuable, in terms of mentoring, to tell a developer what they did right than to tell them what they did wrong.

**Code Duplication**

**Code Reuse**

**Multiple API request (Information caching via Indexed DB)**

**Computed property dependency checks**

**Running code locally/ validating in docker for UX mistakes**

**Constructive Code Reviews - No personal perception**

**Comments  does not mean complaint**

**Use of right data structures**

**External links :-**

[**https://github.com/ryanmcdermott/clean-code-javascript**](https://github.com/ryanmcdermott/clean-code-javascript)  
  
**Note:**  
  
**Exception**  
The above link also explains about using searchable names <https://github.com/ryanmcdermott/clean-code-javascript#use-searchable-names>. However, it is only to be used for functions that are very small and are about to popped from the execution stack. The reason behind this one is that, declaring a searchable constants(or even a variable) will get occupy a memory which only gets cleared when the function finishes execution. Therefore, function which are complex and long should never have these searchable constants.