




 Secrets


 ABAP


 Apex


 C


 C++


 CloudFormation


 COBOL


 C#


 CSS


 Flex


 Go


 HTML


 Java


 **JavaScript**


 Kotlin


 Objective C


 PHP


 PL/I


 PL/SQL


 Python


 RPG


 Ruby


 Scala


 Swift


 Terraform


 Text


 TypeScript

 T-SQL

 VB.NET

 VB6


 XML





JavaScript static code analysis


Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your JAVASCRIPT code


All rules285

 Vulnerability29

 Bug62


 Security Hotspot43

 Code Smell151


 Quick Fix41

Tags ▾


Search by name... 🔍

 Code Smell


"switch" statements should not be nested




Cyclomatic Complexity of functions should not be too high




"strict" mode should be used with caution




Control flow statements "if", "for", "while", "switch" and "try" should not be nested too deeply




"switch" statements should have "default" clauses




"if ... else if" constructs should end with "else" clauses




Control structures should use curly braces




String literals should not be duplicated




Expressions should not be too complex



Local storage should not be used





Template literal placeholder syntax should not be used in regular strings




Template literals should not be nested

Analyze your code

 Code Smell

 Major ?

 brain-overload

confusing

Template literals (previously named "template strings") are an elegant way to build a string without using the + operator to make strings concatenation more readable.

However, it's possible to build complex string literals by nesting together multiple template literals, and therefore lose readability and maintainability.

In such situations, it's preferable to move the nested template into a separate statement.

Noncompliant Code Example

```
let color = "red";
let count = 3;
let message = `I have ${color ? `${count} ${color}` : count}`;
```

Compliant Solution

```
let color = "red";
let count = 3;
let apples = color ? `${count} ${color}` : count;
let message = `I have ${apples} apples`;
```

Available In:

sonarlint

sonarcloud





sonarqube

© 2008-2022 SonarSource S.A., Switzerland. All content is copyright protected. SONAR, SONARSOURCE, SONARLINT, SONARQUBE and SONARCLOUD are trademarks of SonarSource S.A. All other trademarks and copyrights are the property of their respective owners. All rights are expressly reserved.

[Privacy Policy](#)

https://rules.sonarsource.com/javascript/RSPEC-4624

1/2

built-in objects should not be overridden  Bug
"for...in" loops should filter properties before acting on them  Bug
Results of operations on strings should not be ignored  Bug
Increment (++) and decrement (--) operators should not be used in a method call or mixed with other operators in an expression  Code Smell