




 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



## TypeScript static code analysis

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

All rules 279

Vulnerability 27

Bug 51


Security Hotspot 43

Code Smell 158


Quick Fix 50

Tags ▾


Search by name... 🔍

 Code Smell


Primitive types should be omitted from initialized or defaulted declarations




Non-null assertions should not be used




"undefined" should not be assigned




Trailing commas should not be used




Array constructors should not be used




Quotes for string literals should be used consistently




Statements should end with semicolons




Comments should not be located at the end of lines of code




Loops should not contain more than a single "break" or "continue" statement



Variable, property and parameter names should comply with a naming convention






Lines should not end with trailing whitespaces



### Regular expression literals should be used when possible

Analyze your code

 Code Smell  Minor ? Quick Fix ?  regex

Regular expression literals should be preferred over the `RegExp` constructor calls when the pattern is a literal. Simply using a regular expression literal is more concise and easier to read and does not require escaping like a string literal does.




Using the `RegExp` constructor is suitable when the pattern is computed dynamically, e.g. when it is provided by the user.

#### Noncompliant Code Example

```
new RegExp(/foo/);
new RegExp('bar');
new RegExp('baz', 'i');
new RegExp("\\d+");
new RegExp(`qux|quuz`);
```

#### Compliant Solution

```
/foo/;
/bar/;
/baz/i;
/\d+/;
/qux|quuz/;
new RegExp(`Dear ${title},`);
```

Available In:  
  

© 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/typescript/RSPEC-6325

1/2

<div>Files should contain an empty newline at the end</div> <div> Code Smell</div>
<div>An open curly brace should be located at the end of a line</div> <div> Code Smell</div>
<div>Tabulation characters should not be used</div> <div> Code Smell</div>
<div>Function and method names should comply with a naming convention</div> <div> Code Smell</div>