|  |
| --- |
| nest-burger-builder  Version not provided  Code ANALYSIS |

|  |
| --- |
| By: default  2019-06-07 |

# Content

[Content 1](#_Toc492382915)

[Introduction 2](#_Toc492382916)

[Configuration 2](#_Toc492382917)

[Synthesis 2](#_Toc492382918)

[Volume 2](#_Toc492382919)

[Issues count by severity and type 2](#_Toc492382920)

[Charts 2](#_Toc492382921)

[Issues 3](#_Toc492382922)

# Introduction

This document contains results of the code analysis of nest-burger-builder.

# Configuration

* Quality Profiles
  + Names: super-strict [TypeScript];
  + Files: AWq7R1nJgA0evsCbwQeF.json;
* Quality Gate
  + Name: super-strict
  + File: super-strict.xml

# Synthesis

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Quality Gate | Reliability | Security | Maintainability | Coverage | Duplication |
| OK | A | A | A | 82.8 % | 0.0 % |

# Volume

|  |  |
| --- | --- |
| Language | Number |
| TypeScript | 329 |
| Total | 329 |

# Issues count by severity and type

|  |  |  |
| --- | --- | --- |
| Type | Severity | Number |
| VULNERABILITY | BLOCKER | 0 |
| VULNERABILITY | CRITICAL | 0 |
| VULNERABILITY | MAJOR | 0 |
| VULNERABILITY | MINOR | 0 |
| VULNERABILITY | INFO | 0 |
| BUG | BLOCKER | 0 |
| BUG | CRITICAL | 0 |
| BUG | MAJOR | 0 |
| BUG | MINOR | 0 |
| BUG | INFO | 0 |
| CODE\_SMELL | BLOCKER | 0 |
| CODE\_SMELL | CRITICAL | 0 |
| CODE\_SMELL | MAJOR | 10 |
| CODE\_SMELL | MINOR | 0 |
| CODE\_SMELL | INFO | 0 |

# Charts

# Issues

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Description | Type | Severity | Number |
| The "any" type should not be used | Variables can be declared with or without types. Variables declared without a type will be implicitly typed if the declaration includes an initialization, and compiler type checking will be automatically applied to any typed variable. But if you declare a variable with the any "type" then you've explicitly told the compiler not to do any type checking, which is risky. Noncompliant Code Example let a = 42; // implicitly typed to number let b: number = 42; // explicitly typed to number let c: any = 42; // Noncompliant Compliant Solution let a = 42; let b: number = 42; let c: number = 42; | CODE\_SMELL | MAJOR | 8 |
| Magic numbers should not be used | A magic number is a number that comes out of nowhere, and is directly used in a statement. Magic numbers are often used, for instance to limit the number of iterations of a loops, to test the value of a property, etc. Using magic numbers may seem obvious and straightforward when you're writing a piece of code, but they are much less obvious and straightforward at debugging time. That is why magic numbers must be demystified by first being assigned to clearly named variables before being used. -1, 0 and 1 are not considered magic numbers. Noncompliant Code Example function doSomething() { for (let i = 0; i &lt; 4; i++) { // Noncompliant, 4 is a magic number // ... } } Compliant Solution function doSomething() { const numberOfCycles = 4; for (let i = 0; i &lt; numberOfCycles; i++) { // ... } } | CODE\_SMELL | MAJOR | 1 |
| Lines should have sufficient coverage by tests | An issue is created on a file as soon as the line coverage on this file is less than the required threshold. It gives the number of lines to be covered in order to reach the required threshold. | CODE\_SMELL | MAJOR | 1 |