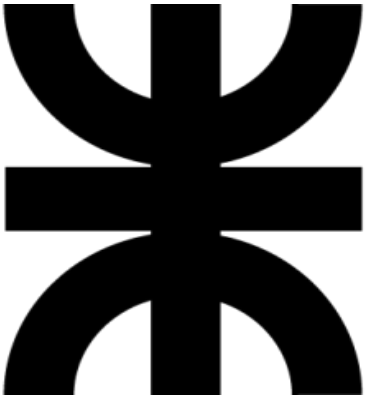

UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL DE CÓRDOBA
Ingeniería en Sistemas de Información

Cátedra Ingeniería y Calidad de Software
Comisión 4k4

Trabajo Práctico N°6

Implementación de User Stories
Documento de Estilo de Código

Grupo 4

Integrantes:

- **Décimo, Sofía Mailén - 89401**
- **Fuentes, Matías - 90463**
- **Gregorat, Franco Lautaro - 89882**
- **Ibarguren, Carlos - 93911**
- **Márquez, Juan Francisco - 89074**
- **Mizzau Anadón, Federico Agustín - 89542**
- **Silvestre, Exequiel - 81811**
- **Toledo, Antonio Ramon - 81864**

Docentes:

- **Ing. Mickaela Crespo**
- **Ing. Georgina González**
- **Constanza Garnero**

Tecnologías usadas en la implementación

Para la implementación de la User story, decidimos utilizar stack de tecnologías para Aplicaciones web responsive, es decir, con una versión mobile:

- Lenguaje de programación: **Javascript**
- Frameworks de trabajo: **React con Vite.**
- Librerías utilizadas: **Eslintrc** para la aplicación de las reglas de estilo de código; **MaterialUI** para la interfaz de usuario; **Email.JS** para el envío de mails.

Reglas de Estilo de Código - JavaScript Standard Style

Documento Oficial: https://standardjs.com/rules

- Usar **2 espacios como sangría.**
  Regla: eslint:indent

```
function hello (name) {
  console.log('hi', name)
}
```

- **Usar comillas simples en cadenas de texto** con la excepción de evitar escapado de texto.
  Regla: eslint:quotes

```
console.log('hello there')    // ✓ ok
console.log("hello there")    // ✗ avoid
console.log(`hello there`)    // ✗ avoid

$("<div class='box'>")        // ✓ ok
console.log(`hello ${name}`)  // ✓ ok
```

- No dejar **variables sin usar**.
  Regla: eslint:no-unused-vars

```
function myFunction () {
  var result = something()   // ✗ avoid
}
```

- Agregue un **espacio después de las palabras clave**.
  Regla: eslint:keyword-spacing

```
if (condition) { ... }   // ✓ ok
if(condition) { ... }    // ✗ avoid
```

- Agregar un espacio antes de los paréntesis de una declaración de función.
  Regla: eslint:space-before-function-paren

```
function name (arg) { ... }   // ✓ ok
function name(arg) { ... }    // ✗ avoid

run(function () { ... })      // ✓ ok
run(function() { ... })       // ✗ avoid
```

- Utilice siempre === en lugar de ==.
  Excepción: obj == nullse permite verificar null || undefined.
  Regla: eslint:eqeqeq

```
if (name === 'John')   // ✓ ok
if (name == 'John')    // ✗ avoid
```

```
if (name !== 'John')   // ✓ ok
if (name != 'John')    // ✗ avoid
```

- Los **operadores infijos** deben estar espaciados.
  Regla: eslint:space-infix-ops

```
// ✓ ok
var x = 2
var message = 'hello, ' + name + '!'
```

```
// ✗ avoid
var x=2
var message = 'hello, '+name+'!'
```

- Las **comas deben tener un espacio** después de ellas.
  Regla: eslint:comma-spacing

```
// ✓ ok
var list = [1, 2, 3, 4]
function greet (name, options) { ... }
```

```
// X avoid
var list = [1,2,3,4]
function greet (name,options) { ... }
```

- Mantenga **las declaraciones else en la misma línea que sus llaves**.
  Regla: eslint:brace-style

```
// ✓ ok
if (condition) {
  // ...
} else {
  // ...
}
```

```
// X avoid
if (condition) {
  // ...
}
else {
  // ...
}
```

- Para **declaraciones if de varias líneas**, utilice llaves.
  Regla: eslint:curly

```
// ✓ ok
if (options.quiet !== true) console.log('done')
```

```
// ✓ ok
if (options.quiet !== true) {
  console.log('done')
}
```

```
// X avoid
if (options.quiet !== true)
  console.log('done')
```

- Gestionar siempre el **parámetro err** en las funciones.
  Regla: eslint:handle-callback-err

```
// ✓ ok
run(function (err) {
  if (err) throw err
  window.alert('done')
})
```

```
// X avoid
run(function (err) {
  window.alert('done')
})
```

- Declarar las **variables globales del navegador** con un comentario **/* global */**. *Las excepciones son: **window**, **document** y **navigator**.*
  *Evita el uso accidental de elementos globales del navegador con nombres incorrectos, como **open**, **length**, **event** y **name**.*

```
/* global alert, prompt */

alert('hi')
prompt('ok?')
```

  También está bien hacer referencia explícita a la función o propiedad **window**, aunque dicho código no se ejecutará en un Worker que use **self** en lugar de **window**.
  Regla: eslint:no-undef

```
window.alert('hi')   // ✓ ok
```

- No se permiten **varias líneas en blanco**.
  Regla: eslint:no-multiple-empty-lines

```
// ✓ ok
var value = 'hello world'
console.log(value)
```

```
// X avoid
var value = 'hello world'
// blank line
// blank line
console.log(value)
```

- Para el operador ternario en una configuración de varias líneas, colocar ? y : en líneas individuales.
  Regla: eslint:operator-linebreak

```
// ✓ ok
var location = env.development ? 'localhost' : 'www.api.com'

// ✓ ok
var location = env.development
  ? 'localhost'
  : 'www.api.com'

// ✗ avoid
var location = env.development ?
  'localhost' :
  'www.api.com'
```

- Para **declaraciones var**, escriba **cada declaración por separado**.
  Regla: eslint:one-var

```
// ✓ ok
var silent = true
var verbose = true

// ✗ avoid
var silent = true, verbose = true

// ✗ avoid
var silent = true,
    verbose = true
```

- Envuelva las **asignaciones condicionales con paréntesis adicionales**. Esto deja claro que **la expresión es intencionalmente una asignación (=)** en lugar de un operador de igualdad (===).
  Regla: eslint:no-cond-assign

```
// ✓ ok
while ((m = text.match(expr))) {
  // ...
}

// ✗ avoid
while (m = text.match(expr)) {
  // ...
}
```

- Agregue espacios dentro de bloques de una sola línea.
  Regla: eslint:block-spacing

```
function foo () {return true}    // ✗ avoid
function foo () { return true }  // ✓ ok
```

- Utilice **camelcase** al **nombrar variables y funciones**.
  Regla: eslint:camelcase

```
function my_function () { }    // ✗ avoid
function myFunction () { }     // ✓ ok

var my_var = 'hello'           // ✗ avoid
var myVar = 'hello'            // ✓ ok
```

- No se permiten **comas al final**.
  Regla: eslint:comma-dangle

```
var obj = {
  message: 'hello',   // ✗ avoid
}
```

- Se deben colocar las **comas al final de la línea**.
  Regla: eslint:comma-style

```
var obj = {
  foo: 'foo'
  ,bar: 'bar'   // ✗ avoid
}

var obj = {
  foo: 'foo',
  bar: 'bar'   // ✓ ok
}
```

- El **punto** debe estar en la **misma línea que la propiedad**.
  Regla: eslint:dot-location

```
console.
  log('hello')  // ✗ avoid

console
  .log('hello') // ✓ ok
```

- Los **archivos deben terminar** con una **nueva línea**.
  Regla: eslint:eol-last

- No debe haber **espacios entre los identificadores de funciones** y sus invocaciones.
  Regla: eslint:func-call-spacing

```
console.log ('hello') // ✗ avoid
console.log('hello')  // ✓ ok
```

- Agregar **espacio entre dos puntos y el valor en pares clave-valor.**
  Regla: eslint:key-spacing

```
var obj = { 'key' : 'value' }    // ✗ avoid
var obj = { 'key' :'value' }     // ✗ avoid
var obj = { 'key':'value' }      // ✗ avoid
var obj = { 'key': 'value' }     // ✓ ok
```

- Los **nombres de los constructores** deben comenzar con **letra mayúscula**.
  Regla: eslint:new-cap

```
function animal () {}
var dog = new animal()    // X avoid

function Animal () {}
var dog = new Animal()    // ✓ ok
```

- El **constructor vacío (sin argumentos)** debe invocarse entre paréntesis.
  Regla: eslint:new-parens

```
function Animal () {}
var dog = new Animal    // X avoid
var dog = new Animal()  // ✓ ok
```

- Los objetos **deben contener un getter** cuando **se define un setter**.
  Regla: eslint:accessor-pairs

```
var person = {
  set name (value) {    // X avoid
    this._name = value
  }
}

var person = {
  set name (value) {
    this._name = value
  },
  get name () {         // ✓ ok
    return this._name
  }
}
```

- Los constructores de clases derivadas **deben ejecutar super**.
  Regla: eslint:constructor-super

```
class Dog {
  constructor () {
    super()             // X avoid
    this.legs = 4
  }
}

class Dog extends Animal {
  constructor () {      // X avoid
    this.legs = 4
  }
}

class Dog extends Animal {
  constructor () {
    super()             // ✓ ok
    this.legs = 4
  }
}
```

- Utilizar **arreglos literales** en lugar del **constructor de arreglos**.
  Regla: eslint:no-array-constructor

```
var nums = new Array(1, 2, 3)   // X avoid
var nums = [1, 2, 3]            // ✓ ok
```

- Evitar el uso de **arguments.callee** y **arguments.caller**.
  Regla: eslint:no-caller

```
function foo (n) {
  if (n <= 0) return

  arguments.callee(n - 1)   // X avoid
}

function foo (n) {
  if (n <= 0) return

  foo(n - 1)                // ✓ ok
}
```

- Evite modificar variables de declaraciones de clases.
  Regla: eslint:no-class-assign

```
class Dog {}
Dog = 'Fido'    // X avoid
```

- **Evite modificar** las **variables** declaradas mediante **const.**
  Regla: eslint:no-const-assign

```
const score = 100
score = 125       // X avoid
```

- Evite el uso de **expresiones constantes** en **condiciones** (excepto bucles).
  Regla: eslint:no-constant-condition

```
if (false) {    // X avoid
  // ...
}

if (x === 0) {  // ✓ ok
  // ...
}

while (true) {  // ✓ ok
  // ...
}
```

- No usar **caracteres de control** en **expresiones regulares**.
  Regla: eslint:no-control-regex

```
var pattern = /\x1f/    // X avoid
var pattern = /\x20/    // ✓ ok
```

- No usar declaraciones ***debugger***.
  Regla: eslint:no-debugger

```
function sum (a, b) {
  debugger      // X avoid
  return a + b
}
```

- No usar el operador **delete** en variables.
  Regla: eslint:no-delete-var

```
var name
delete name     // ✗ avoid
```

- **No duplicar argumentos** en las definiciones de **funciones**.
  Regla: eslint:no-dupe-args

```
function sum (a, b, a) {  // ✗ avoid
  // ...
}

function sum (a, b, c) {  // ✓ ok
  // ...
}
```

- No **duplicar nombres** entre los **miembros de una clase**.
  Regla: eslint:no-dupe-class-members

```
class Dog {
  bark () {}
  bark () {}    // ✗ avoid
}
```

- **No duplicar claves** en los **objetos**.
  Regla: eslint:no-dupe-keys

```
var user = {
  name: 'Jane Doe',
  name: 'John Doe'    // ✗ avoid
}
```

- **No duplicar etiquetas case** en las declaraciones **switch**.
  Regla: eslint:no-duplicate-case

```
switch (id) {
  case 1:
    // ...
  case 1:     // ✗ avoid
}
```

- Utilice **una única declaración import** por **cada módulo**.
  Regla: eslint:no-duplicate-imports

```
import { myFunc1 } from 'module'
import { myFunc2 } from 'module'          // X avoid

import { myFunc1, myFunc2 } from 'module' // ✓ ok
```

- No incluir **caracteres vacíos** en **expresiones regulares**.
  Regla: eslint:no-empty-character-class

```
const myRegex = /^abc[]/      // X avoid
const myRegex = /^abc[a-z]/   // ✓ ok
```

- No usar **patrones desestructurantes vacíos**.
  Regla: eslint:no-empty-pattern

```
const { a: {} } = foo         // X avoid
const { a: { b } } = foo      // ✓ ok
```

- **No usar** la función **eval().**
  Regla: eslint:no-eval

```
eval( "var result = user." + propName ) // X avoid
var result = user[propName]             // ✓ ok
```

- **No reasignar expresiones** en **cláusulas catch**.
  Regla: eslint:no-ex-assign

```
try {
  // ...
} catch (e) {
  e = 'new value'             // X avoid
}

try {
  // ...
} catch (e) {
  const newVal = 'new value'  // ✓ ok
}
```

- No extender objetos nativos.
  Regla: eslint:no-extend-native

```
Object.prototype.age = 21     // X avoid
```

- **Evitar vinculación** de funciones **innecesarias**.
  Regla: eslint:no-extra-bind

```
const name = function () {
  getName()
}.bind(user)    // X avoid

const name = function () {
  this.getName()
}.bind(user)    // ✓ ok
```

- Evitar conversiones booleanas innecesarias.
  Regla: eslint:no-extra-boolean-cast

```
const result = true
if (!!result) {   // X avoid
  // ...
}

const result = true
if (result) {     // ✓ ok
  // ...
}
```

- **No usar paréntesis innecesarios** alrededor de las **expresiones de función**.
  Regla: eslint:no-extra-parens

```
const myFunc = (function () { })   // X avoid
const myFunc = function () { }     // ✓ ok
```

- Usar **break()** para **evitar la ejecución en cascada** de las **cláusulas case**.
  Regla: eslint:no-fallthrough

```
switch (filter) {
  case 1:
    doSomething()    // X avoid
  case 2:
    doSomethingElse()
}

switch (filter) {
  case 1:
    doSomething()
    break           // ✓ ok
  case 2:
    doSomethingElse()
}

switch (filter) {
  case 1:
    doSomething()
    // fallthrough  // ✓ ok
  case 2:
    doSomethingElse()
}
```

- No usar decimales flotantes.
  Regla: eslint:no-floating-decimal

```
const discount = .5      // X avoid
const discount = 0.5     // ✓ ok
```

- **Evitar reasignar** declaraciones de **funciones**.
  Regla: eslint:no-func-assign

```
function myFunc () { }
myFunc = myOtherFunc    // X avoid
```

- **No reasignar variables globales** de **solo lectura**.
  Regla: eslint:no-global-assign

```
window = {}     // X avoid
```

- No usar la función **eval()** de **forma implícita.**
  Regla: eslint:no-implied-eval

```
setTimeout("alert('Hello world')")                  // X avoid
setTimeout(function () { alert('Hello world') })    // ✓ ok
```

- **No declarar funciones** en **bloques anidados**.
  Regla: eslint:no-inner-declarations

```
if (authenticated) {
  function setAuthUser () {}    // X avoid
}
```

- **Evitar cadenas de expresiones regulares inválidas** en los constructores **RegExp**.
  Regla: eslint:no-invalid-regexp

```
RegExp('[a-z')    // X avoid
RegExp('[a-z]')   // ✓ ok
```

- No usar espacios en blanco irregulares.
  Regla: eslint:no-irregular-whitespace

```
function myFunc () /*<NBSP>*/{}   // X avoid
```

- Evitar el uso de __iterator__.
  Regla: eslint:no-iterator

```
Foo.prototype.__iterator__ = function () {}   // X avoid
```

- No usar etiquetas que compartan un nombre con una variable en el ámbito de un bloque.
  Regla: eslint:no-label-var

```
var score = 100
function game () {
  score: while (true) {      // X avoid
    score -= 10
    if (score > 0) continue score
    break
  }
}
```

- No usar declaraciones ***label***.
  Regla: eslint:no-labels

```
label:
  while (true) {
    break label     // X avoid
  }
```

- No usar bloques anidados innecesarios.
  Regla: eslint:no-lone-blocks

```
function myFunc () {
  {                 // X avoid
    myOtherFunc()
  }
}

function myFunc () {
  myOtherFunc()     // ✓ ok
}
```

- **Evite mezclar espacios y tabulaciones** para la indentación.
  Regla: eslint:no-mixed-spaces-and-tabs

- **No usar múltiples espacios** excepto para indentación.
  Regla: eslint:no-multi-spaces

```
const id =    1234    // X avoid
const id = 1234       // ✓ ok
```

- **Evitar** las **cadenas multilínea**.
  Regla: eslint:no-multi-str

```
const message = 'Hello \
                world'     // X avoid
```

- No usar **new sin asignar el objeto** a una variable.
  Regla: eslint:no-new

```
new Character()                   // X avoid
const character = new Character()   // ✓ ok
```

- **No usar** el **constructor *Function***.
  Regla: eslint:no-new-func

```
var sum = new Function('a', 'b', 'return a + b')    // X avoid
```

- **No usar** el **constructor *Object***.
  Regla: eslint:no-new-object

```
let config = new Object()   // X avoid
```

- **No usar** la declaración ***new require***.
  Regla: eslint:no-new-require

```
const myModule = new require('my-module')    // X avoid
```

- No usar el constructor ***Symbol***.
  Regla: eslint:no-new-symbol

```
const foo = new Symbol('foo')   // X avoid
```

- No utilizar **instancias contenedoras primitivas**.
  Regla: eslint:no-new-wrappers

```
const message = new String('hello')   // X avoid
```

- **No llamar a propiedades de objetos globales** como funciones.
  Regla: eslint:no-obj-calls

```
const math = Math()   // X avoid
```

- No usar literales octales.
  Regla: eslint:no-octal

```
const octal = 042          // X avoid
const decimal = 34         // ✓ ok
const octalString = '042' // ✓ ok
```

- No usar secuencias de escape octales en cadenas literales.
  Regla: eslint:no-octal-escape

```
const copyright = 'Copyright \251'  // ✗ avoid
```

- Evite la concatenación de cadenas cuando utilice **__dirname** y **__filename**.
  Regla: eslint:no-path-concat

```
const pathToFile = __dirname + '/app.js'            // ✗ avoid
const pathToFile = path.join(__dirname, 'app.js')   // ✓ ok
```

- Evitar el uso de **__proto__**. Usar **getPrototypeOf()** en su lugar.
  Regla: eslint:no-proto

```
const foo = obj.__proto__               // ✗ avoid
const foo = Object.getPrototypeOf(obj)  // ✓ ok
```

- **Evitar redeclarar variables**.
  Regla: eslint:no-redeclare

```
let name = 'John'
let name = 'Jane'     // ✗ avoid

let name = 'John'
name = 'Jane'         // ✓ ok
```

- Evitar **múltiples espacios** en literales de **expresiones regulares**.
  Regla: eslint:no-regex-spaces

```
const regexp = /test   value/   // ✗ avoid

const regexp = /test {3}value/  // ✓ ok
const regexp = /test value/     // ✓ ok
```

- Las **asignaciones en las declaraciones return** deben estar **entre paréntesis**.
  Regla: eslint:no-return-assign

```
function sum (a, b) {
  return result = a + b     // ✗ avoid
}

function sum (a, b) {
  return (result = a + b)   // ✓ ok
}
```

- Evite asignar una variable a sí misma.
  Regla: eslint:no-self-assign

```
name = name   // X avoid
```

- Evite **comparar una variable consigo misma**.
  Regla: eslint:no-self-compare

```
if (score === score) {}   // X avoid
```

- **Evite** utilizar el **operador de coma.**
  Regla: eslint:no-sequences

```
if (doSomething(), !!test) {}   // X avoid
```

- Los **nombres reservados** no deben ocultarse.
  Regla: eslint:no-shadow-restricted-names

```
let undefined = 'value'     // X avoid
```

- No se permiten **arreglos dispersos.**
  Regla: eslint:no-sparse-arrays

```
let fruits = ['apple',, 'orange']       // X avoid
```

- No se deben utilizar **tabulaciones**.
  Regla: eslint:no-tabs

- Las **cadenas regulares** no deben contener literales de plantilla (template literals).
  Regla: eslint:no-template-curly-in-string

```
const message = 'Hello ${name}'   // X avoid
const message = `Hello ${name}`   // ✓ ok
```

- Se debe **usar super()** antes de **llamar a this**.
  Regla: eslint:no-this-before-super

```
class Dog extends Animal {
  constructor () {
    this.legs = 4     // X avoid
    super()
  }
}
```

- Sólo **invocar un objeto Error** con **throw**.
  Regla: eslint:no-throw-literal

```
throw 'error'               // X avoid
throw new Error('error')    // ✓ ok
```

- No usar espacios en blanco al final de la línea.
  Regla: eslint:no-trailing-spaces

- No inicializar undefined.
  Regla: eslint:no-undef-init

```
let name = undefined    // X avoid

let name
name = 'value'          // ✓ ok
```

- No usar **condiciones no-modificadas** en bucles.
  Regla: eslint:no-unmodified-loop-condition

```
for (let i = 0; i < items.length; j++) {...}    // X avoid
for (let i = 0; i < items.length; i++) {...}    // ✓ ok
```

- **Evitar operadores ternarios** cuando **existen alternativas más simples**.
  Regla: eslint:no-unneeded-ternary

```
let score = val ? val : 0     // X avoid
let score = val || 0          // ✓ ok
```

- No incluir código inalcanzable después de las declaraciones ***return***, ***throw***, ***continue*** y ***break***.
  Regla: eslint:no-unreachable

```
function doSomething () {
  return true
  console.log('never called')     // X avoid
}
```

- No usar **declaraciones de control de flujo** en **bloques finally**.
  Regla: eslint:no-unsafe-finally

```
try {
  // ...
} catch (e) {
  // ...
} finally {
  return 42     // X avoid
}
```

- El **operando izquierdo** de los operadores relacionales **no se debe negar**.
  Regla: eslint:no-unsafe-negation

```
if (!key in obj) {}       // X avoid
if (!(key in obj)) {}     // ✓ ok
```

- Evite el uso innecesario de **.call()** y **.apply()**.
  Regla: eslint:no-useless-call

```
sum.call(null, 1, 2, 3)   // X avoid
```

- Evite el uso de **propiedades computadas innecesarias** en los objetos.
  Regla: eslint:no-useless-computed-key

```
const user = { ['name']: 'John Doe' }   // X avoid
const user = { name: 'John Doe' }       // ✓ ok
```

- No usar innecesariamente constructores.
  Regla: eslint:no-useless-constructor

```
class Car {
  constructor () {      // X avoid
  }
}
```

- No usar escapes innecesarios.
  Regla: eslint:no-useless-escape

```
let message = 'Hell\o'  // X avoid
```

- No se permite **cambiar el nombre** de las asignaciones de **importación, exportación y desestructuración** al mismo nombre.
  Regla: eslint:no-useless-rename

```
import { config as config } from './config'      // X avoid
import { config } from './config'                // ✓ ok
```

- No incluir **espacios en blanco** antes de las propiedades.
  Regla: eslint:no-whitespace-before-property

```
user .name      // X avoid
user.name       // ✓ ok
```

- No usar declaraciones ***with***.
  Regla: eslint:no-with

```
with (val) {...}    // X avoid
```

- Mantenga la **coherencia de las nuevas líneas** entre las **propiedades del objeto**.
  Regla: eslint:object-property-newline

```
const user = {
  name: 'Jane Doe', age: 30,
  username: 'jdoe86'            // X avoid
}

const user = { name: 'Jane Doe', age: 30, username: 'jdoe86' }    // ✓ ok

const user = {
  name: 'Jane Doe',
  age: 30,
  username: 'jdoe86'
}                                                                 // ✓ ok
```

- **Evitar el relleno** dentro de los **bloques.**
  Regla: eslint:padded-blocks

```
if (user) {
                              // X avoid
  const name = getName()

}

if (user) {
  const name = getName()     // ✓ ok
}
```

- Evitar espacios en blanco entre los operadores de extensión y sus expresiones.
  Regla: eslint:rest-spread-spacing

```
fn(... args)    // X avoid
fn(...args)     // ✓ ok
```

- El **punto y coma**, al usarse, debe tener **un espacio después y ningún espacio antes**.
  Regla: eslint:semi-spacing

```
for (let i = 0 ;i < items.length ;i++) {...}    // X avoid
for (let i = 0; i < items.length; i++) {...}    // ✓ ok
```

- Se debe incluir un espacio antes de los bloques.
  Regla: eslint:space-before-blocks

```
if (admin){...}     // X avoid
if (admin) {...}    // ✓ ok
```

- **Evitar** los **espacios dentro de paréntesis**.
  Regla: eslint:space-in-parens

```
getName( name )     // X avoid
getName(name)       // ✓ ok
```

- Los **operadores unarios** deben tener un **espacio después**.
  Regla: eslint:space-unary-ops

```
typeof!admin        // X avoid
typeof !admin        // ✓ ok
```

- Utilice **espacios dentro de los comentarios.**
  Regla: eslint:spaced-comment

```
//comment           // X avoid
// comment          // ✓ ok

/*comment*/         // X avoid
/* comment */       // ✓ ok
```

- **Evitar los espacios** en las **cadenas de plantilla**.
  Regla: eslint:template-curly-spacing

```
const message = `Hello, ${ name }`    // X avoid
const message = `Hello, ${name}`      // ✓ ok
```

- Usar **isNaN()** al verificar valores **NaN**.
  Regla: eslint:use-isnan

```
if (price === NaN) { }      // X avoid
if (isNaN(price)) { }       // ✓ ok
```

- Usar **cadenas válidas** para comparar con **typeof**.
  Regla: eslint:valid-typeof

```
typeof name === 'undefimed'     // X avoid
typeof name === 'undefined'     // ✓ ok
```

- Las **expresiones de función invocadas inmediatamente (IIFE)** deben estar **empaquetadas**.
  Regla: eslint:wrap-iife

```
const getName = function () { }()     // X avoid

const getName = (function () { }())   // ✓ ok
const getName = (function () { })()   // ✓ ok
```

- Los * (asteriscos) en **yield* deben tener un espacio antes y después**.
  Regla: eslint:yield-star-spacing

```
yield* increment()    // X avoid
yield * increment()   // ✓ ok
```

- Evite las **“condiciones de Yoda”**.
  Regla: eslint:yoda

```
if (42 === age) { }    // X avoid
if (age === 42) { }    // ✓ ok
```

- **Evitar** el uso de **punto y coma**. (ver: 1 , 2 , 3)
  Regla: eslint:semi

```
window.alert('hi')   // ✓ ok
window.alert('hi');  // X avoid
```

- Nunca comience una línea con **(**, **[**, `` ` `` u otras posibilidades poco probables. Este es el único problema al omitir el punto y coma y **standard** lo protege de este posible problema.
  (La lista completa es: [, (, `` ` ``, +, *, /, -, ,, .pero la mayoría de ellos nunca aparecerán al comienzo de una línea en código real).
  Regla: eslint:no-unexpected-multiline

```
// ✓ ok
;(function () {
  window.alert('ok')
}())

// X avoid
(function () {
  window.alert('ok')
}())
```

```
// ✓ ok
;[1, 2, 3].forEach(bar)

// X avoid
[1, 2, 3].forEach(bar)
```

```
// ✓ ok
;`hello`.indexOf('o')

// X avoid
`hello`.indexOf('o')
```

Nota: Si escribes código como este con frecuencia, es posible que estés intentando ser demasiado inteligente.
Se desaconsejan las taquigrafías inteligentes, en favor de expresiones claras

y legibles, siempre que sea posible.
En lugar de esto:

```
;[1, 2, 3].forEach(bar)
```

- Es preferible esto:

```
var nums = [1, 2, 3]
nums.forEach(bar)
```