Pruebas en JavaScript con ESLINT.

Tests in JavaScript with ESLINT.

Daniel López Huertas[1]
dalohuer2004@hotmail.com

Simena Dinas[2]
simena.dinas00@usc.edu.co

Universidad Santiago de Cali, Facultad de Ingeniería, Programa de Tecnología en Sistemas (1)
Universidad Santiago de Cali, Facultad de Ingeniería, Programa de Ingeniería de Sistemas (2)

Resumen
En este artículo se expone el uso de la herramienta ESLINT para la corrección de código fuente en JavaScript. Inicialmente se muestra el proceso de instalación y la ejecución de la herramienta, posteriormente se hace la configuración de los parámetros necesarios para ajustar el aplicativo de acuerdo con las preferencias de cada programador, sin embargo, estos componentes pueden presentar algunas variaciones y serán explicadas con el fin de mejorar el entendimiento del proceso. Se desarrollarán algunos ejemplos que ayudarán a comprender el uso de ESLINT.

Palabras Clave: ESLINT; configuración; componentes; uso; herramienta.

Abstract
This article describes the use of the ESLINT tool for the correction of source code in JavaScript. Initially, the installation process and the execution of the tool are shown, then the configuration of the necessary parameters is made to adjust the application according to the preferences of each programmer, however, these components can present some variations and will be explained with the in order to improve the understanding of the process. Some examples will be developed that will help to understand the use of ESLINT.

Keywords: ESLINT; setting; components; use; rules; tool.

1. INTRODUCCIÓN

Entre las diferentes metodologías de desarrollo de software existentes lo más común que se puede encontrar en ellas es el uso de etapas de desarrollo, dependiendo de la metodología se pueden establecer las etapas que se usarán. Las etapas más comunes son la de análisis, diseño, desarrollo y pruebas. Para acotar un poco el preámbulo en esta parte, se puede mencionar algunas de las metodologías tales como SCRUM, Rational Unified Process y Extrem Programming, ellas permiten mostrar de distinta manera las etapas de desarrollo. Esto permite a los programadores identificar fácilmente en qué punto se encuentra la ejecución del proyecto de software y conocer si sus avances se ajustan dentro de los plazos fijados en el cronograma de trabajo inicial. Generalmente, en el desarrollo del proyecto de software deben tenerse en cuenta las siguientes etapas: el análisis, el diseño, la implementación y las pruebas (Galiano, 2018). En el análisis suele establecerse con ayuda del cliente o el dueño del producto, todos los requisitos, historias de usuario o documentación que pueda dar a conocer las necesidades del cliente y de ese modo, definir las características más importantes que tendrá el software. Una de las múltiples formas de hacerlo es generando entrevistas entre los clientes y los encargados de diseñar el software, para construir los requisitos del sistema. Luego de tener los requisitos se deben establecer reuniones ocasionales para realizar los afinamientos que haya a lugar en el proceso. En el diseño global del software, por lo general se hace la propuesta de la arquitectura, bases de datos, interfaz, desarrollo de prototipos, entre otras. En algunas de las metodologías de desarrollo de software se inicia a partir de los requisitos de los clientes, por ejemplo, RUP, otras inician a partir de las historias de usuario, por ejemplo, SCRUM y XP. La selección de la metodología adecuada depende de las necesidades y requisitos del proyecto, sin dejar a un lado la importancia de la relación entre los requisitos del análisis y el diseño (Kraffczyk, 2003) (Rodriguez, 2017). En la ejecución es importante escoger las herramientas más idóneas dependiendo de los parámetros del análisis, del diseño, de

las habilidades del usuario, del costo y estabilidad de la herramienta y de la infraestructura de la empresa así de esta forma llevar a cabo una estrategia cuidadosa para diseñar o estructurar todo el código, y no olvidarse de la lógica de cada uno de los bloques a utilizar, que todo sea sencillo y fácil de interpretar y así tener una buena documentación del aplicativo en el interior del programa como en los manuales. Por último, las pruebas que serán el campo de desarrollo en este documento, son cruciales para el progreso de software ya que en ellas se pueden evidenciar errores de sintaxis, de lógica, de tiempo y de estructura, y que cada uno de los inconvenientes que se acaban de mencionar pueden presentarse en cualquiera de las etapas de desarrollo. Es de aclarar que con este proceso se pretende comprobar la calidad del producto y cada vez que se realice una prueba ayude a mejorar más la entrega final. Lo ideal es que todas las fallas puedan detectarse durante la etapa de desarrollo y no en la etapa de producción. Ahora bien, en las pruebas que se puedan realizar se encuentran las pruebas de caja negra que en este caso prueba el aplicativo con valores u opciones fuera de los rangos establecidos y así verificar los limitantes del aplicativo permitiendo realizar los ajustes correspondientes. Por otro lado, se encuentran las pruebas de caja blanca que son dirigidas al interior del código que, por supuesto se encargan de verificar elementos como la sangría o identación, la falta de expresiones o símbolos, la declaración o falta de uso de variables, en general que el código sea limpio y eficiente. Por tal motivo se hará uso de una herramienta que ayuda al programador a seguir buenas prácticas según las normas y reglas establecidas y que puedan aplicarse al proyecto en cuestión. Para la explicación y desarrollo del contenido de este documento se utilizará un *Linter* llamado **ESLINT,** con éste se puede llevar a buen término la ejecución de las pruebas de caja blanca necesarias para verificar el código de cualquier proyecto de software, ahora bien, se debe tener en cuenta que necesita de algunos componentes antes de utilizarlo en la verificación del código, componentes cuyo funcionamiento y forma de trabajo se desarrollarán como parte de la configuración, instalación y utilización de ESLINT. (ESLINT, 2014) (Pozzi, 2017) (Domínguez, 2019)

2. METODOLOGÍA

Para realizar este documento se utilizaron diferentes fuentes y técnicas de recolección de información secundaria, como lo son documentos técnicos, páginas web y artículos de universidades. La mayor fuente de información se obtuvo de páginas web, ya que los desarrolladores mayormente hacen las publicaciones respecto a ESLINT en la web (@fokusman, 2019). Para obtener mejores resultados en cuanto a la calidad del código, se verifican todas las fuentes de información anteriormente citadas.

Respecto a las herramientas utilizadas se descargan las últimas versiones de los aplicativos de internet, ya que ello garantiza una instalación sin errores de compatibilidad, tanto como el NODE.js (Abernethy, 2011), el NPM (Benites, 2017), ESLINT y ATOM son aplicativos de licencia de uso gratuito, por tal motivo cada uno se descarga desde la página web correspondiente. Al descargar e instalar NODE.js ya permite la ejecución de NPM por la consola de Windows, con este paso completado es posible ejecutar el comando de instalación local o global de ESLINT en la consola. Hasta este momento se tiene gran parte instalada de ESLINT, sin embargo, se descarga de la web el instalador de la última versión de ATOM que va a ser el editor de texto a utilizar para verificar el código.

Las configuraciones son indispensables ya que en sí los aplicativos no van a reconocer todas las configuraciones de forma automática, por consiguiente, ya instalado ESLINT se procede a hacer una configuración adicional en la consola de Windows y es por tanto la inicialización del archivo de parametrización de ESLINT. No obstante, no es el único que necesita configuración adicional, ya que ATOM necesita instalar un paquete adicional en su gestor de paquetería.

Instalado y configurado ESLINT bastará con terminar la instalación del paquete de ATOM y eso será lo necesario para visualizar el código y verificar la corrección de errores, sugerencias y cambios de estilo que muestra ESLINT.

Para las diferentes pruebas que se van a realizar con ESLINT se escogen algunos segmentos de código ya que no es necesario exponer muchas líneas de código y así utilizar el aplicativo, para muestra de ello se puede ver en los ejercicios durante toda la parte del desarrollo de las pruebas.

Cuando se usa ESLINT en la verificación aparecen los diferentes hallazgos directamente al inicio de las líneas de código,

esto permite que durante el desarrollo de software se detecten inmediatamente las novedades y fallas que deban corregirse, dando la opción de hacerlo en ese momento o dejar al final para utilizar la opción de verificación y corrección del código.

Utilizar la herramienta ESLINT para mantener limpio el código, permite obtener resultados satisfactorios, tanto para un grupo de trabajo como para una sola persona. Lo anterior radica en que ESLINT organiza el código para que sea entendible por otras personas que necesiten continuar con el trabajo que ya inició otro programador.

3. ESLINT

Es una herramienta entre muchas otras utilizada por los programadores para verificar esquemas, códigos con errores o problemas de sintaxis entre todas las líneas escritas por el programador, esto sirve para tener estándares de escritura de código y así generar compilaciones más limpias y bien estructuradas en JavaScript. Se dispone del aplicativo para hacer este tipo de revisiones ya que el proceso sería muy tedioso para el programador al momento de intentar encontrar errores en código propio o en el de otras personas involucradas en el proyecto, en caso de que las haya; ya que la escritura de código en JavaScript puede tender a errores por las costumbres de cada programador al realizar la escritura de este. Una característica especial es permitir la revisión del código antes de la compilación o la misma ejecución del aplicativo. Gracias a esta forma de hacer las revisiones de código se ahorra tiempo valioso que puede ser usado en otros aspectos del desarrollo del proyecto de software.» (contributors C. J., Migrating from JSCS, 2016) (contributors C. J., Getting Started with ESLint, 2019).

3.1. COMPONENTES NECESARIOS PARA LA INSTALACIÓN DE ESLINT

Antes de iniciar la instalación de ESLINT se debe tener en cuenta algunos elementos necesarios que posibilitan la ejecución de ESLINT; se habla de NODE.js y NPM en lo posible en sus últimas versiones ya que cada vez que se encuentra una nueva publicación de estos aplicativos en las páginas oficiales trae consigo beneficios de actualización o corrección de errores que se traducen en mejores tiempos de ejecución y desaparición de problemas con las versiones anteriores. (CJ, 2018) (Foundation, 2018) ((TSC), 2019).

Node.js es básicamente utilizar el lenguaje de programación de JavaScript, pero del lado del servidor. Fue construido para diseñar aplicaciones en red escalables, es potente y ligero en la entrada y salida (I/O). Node.js es diferente ya que la forma de comunicación con el servidor es distinta a las conexiones tradicionales, cada conexión nueva genera un evento nuevo dentro del motor de Node.js, por otro lado, las conexiones de otras plataformas de parte del servidor diferentes a Node.js hacen una nueva conexión cada vez y con ello generan un consumo de memoria y conexiones concurrentes que se traducen en el aumento de capacidad de servidores, costos de ancho de banda, tráfico de red y almacenamiento.

Ahora bien, también se utiliza NPM (Node Packege Manager) este gestor de paquetes permite agilizar las instalaciones de librerías o paquetes de instalación con tan solo unos pocos comandos, también ayuda a gestionar, adicionar y organizar todas las dependencias o módulos que se puedan necesitar en las instalaciones correspondientes.

3.2. INSTALACIÓN DE ESLINT

En el proceso de instalación se debe dejar claro que ESLINT es multiplataforma, funciona en Linux, MacOs y Windows. Los procesos de instalación en tanto a comandos son similares, pero durante la instalación de paquetes de ESLINT en los diferentes SO (sistemas operativos) cambia por la estructura de los mismo más no por ESLINT en sí (Adekoya, 2019). En Windows las diferentes carpetas que utiliza van a ser distintas a las estructuras de directorios de Linux o MacOS y así sucesivamente. También se pueden hacer instalaciones de forma global Figura 1 o local Figura 2 y que dependiendo de las necesidades del proyecto de software se elige entre una u otra (Charles, 2017); si se escoge la manera global todo lo que se instale en adelante debe de ser así o viceversa. No se debe mezclar instalaciones entre global y local que de manera

inequívoca presentaría problemas más adelante en las ejecuciones de los aplicativos al momento de necesitarlos.

```
C:\Users\Daniel>npm install -g eslint
C:\Users\Daniel\AppData\Roaming\npm\eslint -> C:\Users\Daniel\AppData\Roaming\npm\node_modules\eslint\bin\eslint.js
+ eslint@6.6.0
added 4 packages from 2 contributors, removed 2 packages and updated 20 packages in 32.775s
```

Figura 1. Comando instalación global ESLINT en Windows 10.

```
C:\Users\Daniel>npm install --save-dev eslint
+ eslint@6.6.0
updated 1 package and audited 690 packages in 31.667s
found 0 vulnerabilities
```

Figura 2. Comando instalación local ESLINT en Windows 10.

Es de resaltar que previo a la configuración debe establecerse la forma de instalación: Global o Local.

3.3. CONFIGURACIÓN INICIAL DE ESLINT

Posterior a la instalación se continúa con la configuración inicial de ESLINT ya que en este paso se definen todos los parámetros iniciales para utilizarlo en la revisión de código de los proyectos que se deseen acomodar de acuerdo con las reglas de cada programador. El comando para utilizar es **eslint - -int**, al momento de ejecutar el comando se presentan una serie de preguntas y la forma en que se muestran en consola se puede apreciar en la Figura 3. (Martín, ESLint, 2019)

```
C:\Users\Daniel>eslint --init
? How would you like to use ESLint? To check syntax, find problems, and enforce code style
? What type of modules does your project use? JavaScript modules (import/export)
? Which framework does your project use? React
? Does your project use TypeScript? Yes
? Where does your code run? Browser, Node
? How would you like to define a style for your project? Answer questions about your style
? What format do you want your config file to be in? JavaScript
? What style of indentation do you use? Spaces
? What quotes do you use for strings? Single
? What line endings do you use? Windows
? Do you require semicolons? Yes
```

Figura 3. Respuestas al comando eslint --init de configuración inicial ESLINT en Windows 10.

De forma general estas son las preguntas presentadas por el instalador a cualquier persona que realice una instalación en un equipo de cómputo, en este texto se procede a analizar las que se seleccionaron para la labor y pruebas de este proyecto.

Entre ellas se elige la opción de verificar sintaxis, encontrar problemas y forzar el estilo de código. Esta opción ofrece la forma más completa de tener el conjunto de reglas acomodadas al gusto particular de la persona que se encuentra haciendo la instalación y como resultado de esta elección cuando se realizan las pruebas se muestran los distintos errores de sintaxis, los problemas en el código y los estilos en el proyecto que se está trabajando (Copes, 2018), por ejemplo: identación del código para una mejor visualización de las líneas de código, declaración de variables, y declaración de métodos dentro del proyecto. Por otro lado, se hace la elección del módulo de JavaScript que solo hace referencia a importar los módulos que se necesiten mientras se esté verificando el código. También se debe escoger el framework que se va a utilizar, sin embargo, en esta opción no se obliga a tomar una en particular ya que dentro de las opciones permite escoger ninguna y así la configuración de ESLINT se ajustará a las necesidades y framework del programador.

Si el proyecto que se va a revisar utiliza "TypeScript" o tiene compatibilidad con éste, se recomienda dejarlo activo ya que hace un análisis del código estático de JavaScript y permite que se utilice luego con el compilador de TypeScript para retroalimentación en los problemas y otros cambios en el código (Gonzalez, 2019) (contributors S. O., 2018). Ya que los

proyectos que se puedan revisar pueden ir orientados a su utilización en navegadores o por el contrario en la parte del servidor, en las configuraciones se permite escoger en donde se va a correr el código si en navegadores o en la parte del servidor con Node.js, y si el proyecto involucra ambos lugares de ejecución, también da la opción de escoger ambos y así ESLINT hará una revisión más completa.

Habiendo recorrido diferentes opciones, la misma instalación se encarga de orientar paso a paso y determinar en qué formato se va a construir el archivo de configuración de la instalación mostrando tres alternativas distintas, el programador puede escoger la que mejor se ajuste a su necesidad. Se puede especificar el estilo de identación que se requiera o que guste más a la persona que vaya a hacer las revisiones. En cuanto a las comillas simples o dobles debe seleccionarse la que se ajusta más al proyecto a realizar, ya que, si no se hace la elección correcta ESLINT al momento de hacer las revisiones encontrará y notificará errores donde no los hay o viceversa dependiendo de cada proyecto. También es necesario establecer bajo qué Sistema Operativo (SO) se va a trabajar, ya que como ESLINT es multiplataforma se va a poder utilizar en los sistemas operativos más comunes y esta personalización hace que ESLINT se amolde correctamente en su sistema huésped.

En JavaScript se puede o no hacer uso del "punto y coma" al finalizar algunas líneas de código, entonces es muy importante denotar si la instalación que se va a realizar de ESLINT debe o no hacer el uso del "punto y coma", esto con el fin de no reportar errores innecesarios por el estilo de escritura de código de cada programador. También se debe tener en cuenta si se activa o no esta característica en un grupo de trabajo ya que todos deben estar de acuerdo en si debe o no estar activo.

Todo lo anterior lleva a que ESLINT puede o no mostrar alertas o notificaciones de los posibles errores, estilos y demás, y es importante que al momento de tener la instalación lista se conozca cuáles son las reglas que se usan o no en la configuración, estas reglas se pueden aplicar de manera muy personalizada o por otro lado se pueden utilizar reglas ya predefinidas de empresas o comunidades de desarrollo. A continuación se da a conocer la Tabla 1 que muestra categorías, la descripción y la cantidad de reglas que se encontraban en la página de ESLINT al 02 de junio de 2017 según (Tómasdóttir, Aniche, & van Deursen, 2018) y con ello se puede comprender que son una cantidad significativa de reglas a estudiar para determinar cuáles son las más adecuadas para el proyecto de software al cual se le puede aplicar las revisión de errores mediante el linter (Hoyos, 2018) ESLINT.

Tabla 1. ESLINT Reglas, categorías en orden y descripción desde la documentación de esling.org en junio de 2017

Category	Description	Available rules
Possible Errors	Possible syntax or logic errors in JavaScript code	31
Best Practices	Better ways of doing things to avoid various problems	69
Strict Mode	Strict mode directives	1
Variables	Rules that relate to variable declarations	12
Node.js and CommonJS	For code running in Node.js, or in browsers with CommonJS	10
Stylistic Issues	Stylistic guidelines where rules can be subjective	81
ECMAScript 6	Rules for new features of ES6 (ES2015)	32
Total		**236**

Fuente: Tomada de (Tómasdóttir, Aniche, & van Deursen, 2018)

De acuerdo a la tabla anterior se hace una revisión de las reglas en ESLINT.org (contributors E. C., 2019) y se resumen en la Tabla 2, se puede concluir que han tenido un aumento significativo en algunas categorías y esto se convierte en una mejor corrección de errores al momento de utilizar ESLINT en la revisión de los proyectos de software.

Tabla 2. ESLINT Reglas disponibles a octubre de 2019

Categoría	Reglas Disponibles
Posibles errores sintácticos o lógicos en código JavaScript	74
Mejores Formas de hacer las cosas para evitar varios problemas	152
Directivas de modo estricto	1
Reglas que se refieren a declaraciones de variables	21
Para la ejecución en Node.js o en navegadores con CommonJS	11

Problemas de estilo pueden ser bastante subjetivas	183
Reglas para ES6 (ES2015)	61
Total	**503**

Fuente: Tomada de ESLINT.org (contributors E. C., 2019)

3.4. USO DE ESLINT EN CÓDIGO JAVASCRIPT

Teniendo todos los elementos necesarios para el uso de ESLINT se procede a revisar el código JavaScript de los proyectos que se puedan tener, Lo anterior se puede realizar a través de dos métodos sencillos y sin utilizar herramientas muy complejas: por consola que permite ejecutar el comando de "eslint «nombreArchivo.js»",dar enter y muestra la revisión de errores como resultado en la misma consola, con los errores ya identificados se procede a abrir el archivo en un bloc de notas y se hace la respectiva corrección (Williams, 2019). O mediante un editor de texto especializado en programación o para la lectura correcta de los archivos según su tipo, en este caso se hará uso de un editor de texto con edición multiplataforma llamado ATOM que entre sus bondades ofrece administrador de paquetes incorporado, autocompletado inteligente, navegador de archivos y paneles múltiples. Este editor de texto multiplataforma mediante su modalidad de paquetes permite adicionar uno llamado "Linter-eslint" que cumple la función de utilizar ESLINT dentro del editor sin necesidad de llamar o ejecutar algún comando, solo con tener instalado este paquete automáticamente al abrir algún código de JavaScript él visualiza los errores inmediatamente sobre las líneas de código a revisar, así como se puede apreciar en la Figura 4.

Figura 4. Muestra de errores en el código con ESLINT en ATOM.

En la figura 5 se muestra la salida de los errores de ESLINT en el panel de notificaciones del aplicativo ATOM y las visualiza en las categorías de gravedad (Severity), proveedor (Provider), descripción (Description) y línea (Line). En la sección gravedad se pueden encontrar mensajes de error o advertencia (warning) como se muestra en la Figura 8, en el proveedor se encuentra el nombre del aplicativo que hace la revisión de código, en la descripción se muestra un mensaje corto con el error o advertencia y en la sección Línea se puede identificar la fila y la columna en que se encuentra la alerta que se debe revisar.

Severity	Provider	Description	Line
Error	ESLint	Strings must use singlequote. (quotes)	3:15
Error	ESLint	Missing semicolon. (semi)	5:10
Error	ESLint	'chocolate' is not defined. (no-undef)	8:14
Error	ESLint	Expected indentation of 4 spaces but found 2. (indent)	10:1

Figura 5. Salida de errores por el panel de notificación de ATOM.

En la descripción (Description) aparte del mensaje corto también muestra entre paréntesis los nombres de las reglas de

ESLINT a las que hace referencia y cuyas categorías se encuentran enumeradas en la Tabla 2. Para mayor referencia de todas las reglas que puede tener ESLINT visitar el sitio web en la siguiente referencia (contributors E. C., 2019).

En los ejemplos que se realizaron y por citar algunas reglas se tienen: quotes que hace referencia a el uso de las comillas simples o dobles, semi es la regla que indica que es requerido el uso de punto y coma, no-undef que hace referencia a que no está definida una variable, indent que significa que necesita tabulación, eol-last que requiere una nueva línea al final del código, linebreak-style que hace referencia a imponer un estilo de salto de línea consistente, entre otras como se menciona anteriormente y que se encuentran documentadas en la página de ESLINT.

En la visualización que ofrece el aplicativo ATOM se puede observar que ESLINT cumple con su función correctamente, ahora bien, por consola el proceso es un poco más dispendioso y no tan automatizado, por tal motivo las revisiones de errores con ESLINT se realizan con el editor de texto multiplataforma ATOM.

En la Figura 4 se observa que al situar el cursor sobre los puntos rojos que aparecen al inicio de las líneas de código, se muestra la sugerencia de solución al error y también un botón "FIX" para dar clic y poder corregirlo al instante. No en todas las ocasiones muestra el botón "FIX" ya que algunos errores son de interpretación y ESLINT no dará una sugerencia para la corrección, en algunos casos los errores de token inesperados en la revisión de código con ESLINT se deben a la discrepancia entre el entorno de desarrollo y las capacidades de análisis en ese momento de ESLINT con los cambios continuos de JavaScript ES6 y ES7, como se muestra en la Figura 6.

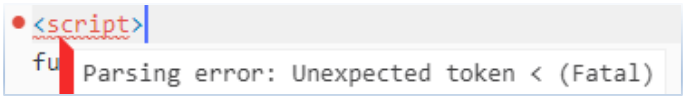
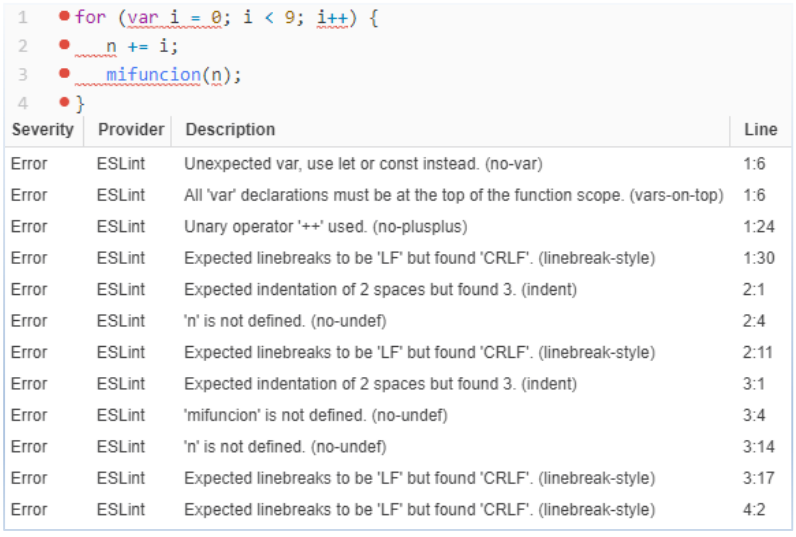


Figura 6. Error que no muestra la opción "FIX".

Ya revisando otro tipo de errores que se pueden encontrar con ESLINT se procede a verificar un ciclo FOR ya que en unas líneas de código cortas se encuentran varias situaciones como las que se muestran en la Figura 7.



```
1  for (var i = 0; i < 9; i++) {
2      n += i;
3      mifuncion(n);
4  }
```

Severity	Provider	Description	Line
Error	ESLint	Unexpected var, use let or const instead. (no-var)	1:6
Error	ESLint	All 'var' declarations must be at the top of the function scope. (vars-on-top)	1:6
Error	ESLint	Unary operator '++' used. (no-plusplus)	1:24
Error	ESLint	Expected linebreaks to be 'LF' but found 'CRLF'. (linebreak-style)	1:30
Error	ESLint	Expected indentation of 2 spaces but found 3. (indent)	2:1
Error	ESLint	'n' is not defined. (no-undef)	2:4
Error	ESLint	Expected linebreaks to be 'LF' but found 'CRLF'. (linebreak-style)	2:11
Error	ESLint	Expected indentation of 2 spaces but found 3. (indent)	3:1
Error	ESLint	'mifuncion' is not defined. (no-undef)	3:4
Error	ESLint	'n' is not defined. (no-undef)	3:14
Error	ESLint	Expected linebreaks to be 'LF' but found 'CRLF'. (linebreak-style)	3:17
Error	ESLint	Expected linebreaks to be 'LF' but found 'CRLF'. (linebreak-style)	4:2

Figura 7. Muestra de errores ciclo FOR con ESLINT.

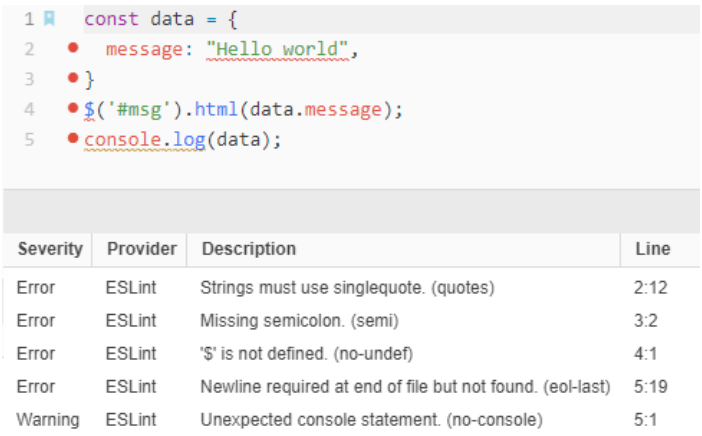


Figura 8. Muestra de errores variados con ESLINT.

4. DISCUSIÓN DE RESULTADOS

En el inicio de la ejecución de ESLINT se pudo evidenciar la solución de errores. Sin embargo, en las pruebas realizadas se alcanza cierto grado de corrección de, errores, sugerencias o ajuste de estilos. En sí, este aplicativo es efectivo hasta donde se pueda configurar, y todo lo pueda detectar y corregir es resultado de ello, pero la realidad se ajusta a otro campo, el de lo expuesto en este documento, lo que manifiesta que no todas las alertas que muestra ESLINT las puede corregir de forma automática sin importar las reglas que se tengan o no activas al momento de hacer la revisión del código. Es de aclarar que la mayoría de las correcciones están dentro de los parámetros permitidos. Sin embargo, es bueno que a nivel personal se pueda hacer otra revisión y estar seguros de que todo está correcto o tal como se requiere. En algunos segmentos de código mostró la alerta de corrección de comillas dobles a simples, al momento de realizar el cambio automático no lo hace correctamente, no obstante, se volvía a colocar el error de forma premeditada y se ejecutaba de nuevo la corrección automática y lo realiza bien, ese hecho infiere que ESLINT necesita ajustes en la forma de cómo abordar y corregir los errores ya que en ocasiones realiza bien las correcciones y en otras no.

Con la cantidad de errores encontrados con ESLINT en la Figura 7 se puede analizar cómo en un código corto se puede corregir de manera sencilla, ahora bien, si el código es más extenso es poco práctico que ESLINT muestre los errores y el programador tenga que hacer las respectivas correcciones una a una. Por consiguiente, cuando se hace la importación del "Linter-eslint" en ATOM no solo le permite revisar los errores, sino que también implementa en el aplicativo ATOM un menú contextual que se llama "ESLint Fix" que aparece al dar clic derecho sobre el código que se está revisando. Ya con este menú no habrá necesidad de corregir manualmente los errores, sino que solo bastará con dar clic sobre el menú y ESLINT hará la corrección de todos los errores automáticamente. Algo muy importante es que no todas las correcciones que realice de forma automática van a ser las correctas y en ese momento es que el programador o revisor del código terminará de hacer los ajustes correspondientes.

Los errores y corrección de estilos dentro del código tienen una gran variedad de matices por así decirlo, son muchas las reglas disponibles, así como se muestra en la Figura 2 y que dependiendo de cuales se activen o no, se pueden obtener los resultados deseados. Es de señalar que todo lo que pueda mostrar ESLINT lo puede hacer como un problema, una sugerencia o una regla de diseño en el código, en todo caso los problemas son los que se deben revisar con premura ya que pueden causar error o mal comportamiento de la ejecución del aplicativo, las sugerencias solo indican la manera en cómo se podría mejorar el código y el diseño solo muestra los espacios de identación, donde van espacios o una nueva línea de código en blanco, y todo lo anterior para que el código revisado quede bien tanto en sintaxis como en diseño. En la Figura 8 se puede apreciar lo mencionado anteriormente.

5. CONCLUSIONES

A pesar de evidenciar errores en la corrección de las alertas y que muestra algunas llamadas sin corregir e incluso las deja marcadas para hacerlo manualmente, se afianza que el buen uso de ESLINT mejora notablemente la forma en que se muestra el código de vista al programador. Las alertas que si puede corregir de forma automática ayudan y disminuyen el tiempo que se pueden tardar los programadores haciendo las pruebas y conlleva a las buenas prácticas en el ámbito de la programación en JavaScript.

Se podría pensar que los problemas de coherencia y errores en código JavaScript tienen una solución definitiva con la herramienta ESLINT como se encuentra estructurada en este momento, pero ya que se está en plena era de la inteligencia artificial se podría pensar en implementarla al núcleo de ESLINT y así no solo pensar en incluir un grupo de reglas ya establecidas, sino implementar en la inteligencia artificial toda la estructura de JavaScript y sus normas para que así pueda fortalecer la forma que ESLINT muestra las alertas y que en la medida del avance las pueda corregir en su totalidad, esto permite una mejor corrección de todo tipo de errores en el código que se pueda revisar y claro está, en la disminución de los errores para el usuario final que reciba el software.

En este documento se puede observar la importancia que pueden tener los aplicativos que se encargan de la revisión de código para los proyectos de software unitarios o grupales, y con la utilización de la herramienta ESLINT se pueden establecer normas o estándares para cumplir dentro de un grupo de desarrollo o en las pruebas unitarias tal cómo se observó dentro del documento. No obstante, cabe resaltar que el utilizar este tipo de programas en la revisión ahorra mucho tiempo ya que las revisiones se realizan antes de la compilación o la fase de producción del software.

REFERENCIAS

(TSC), T. S. (01 de Octubre de 2019). *ESLint*. Obtenido de NPM: https://www.npmjs.com/package/eslint

@fokusman, F. T. (14 de Octubre de 2019). *How linting and ESLint improve code quality*. Obtenido de https://hackernoon.com/: https://hackernoon.com/how-linting-and-eslint-improve-code-quality-fa83d2469efe

Abernethy, M. (14 de 06 de 2011). *¿Simplemente qué es Node.js?* Obtenido de https://www.ibm.com: https://www.ibm.com/developerworks/ssa/opensource/library/os-nodejs/index.html

Adekoya, D. (20 de Febrero de 2019). *Setting up ESLINT in your JavaScript Project with VS Code*. Obtenido de https://dev.to/: https://dev.to/iamdammak/setting-up-eslint-in-your-javascript-project-with-vs-code-2amf

Benites, A. G. (20 de 04 de 2017). *DevCode*. Obtenido de https://devcode.la: https://devcode.la/blog/que-es-npm/

Charles, J. (06 de Enero de 2017). *Better Code Quality with ESLint*. Obtenido de https://www.pluralsight.com/courses/eslint-better-code-quality?utm_source=eslint-dot-org&utm_medium=video&utm_campaign=authordemo#

CJ, C. G. (20 de Febrero de 2018). *How to Install and Configure eslint*. Obtenido de https://www.youtube.com/watch?v=5rd6m1kr5Ig

contributors, C. J. (Abril de 2016). *Migrating from JSCS*. Obtenido de https://eslint.org/: https://eslint.org/docs/user-guide/migrating-from-jscs

contributors, C. J. (15 de Octubre de 2019). *Getting Started with ESLint*. Obtenido de https://eslint.org/: https://eslint.org/docs/user-guide/getting-started

contributors, E. C. (20 de Octubre de 2019). *Rules*. Obtenido de https://eslint.org: https://eslint.org/docs/rules/

contributors, S. O. (10 de septiembre de 2018). *Aprendizaje TypeScript*. Obtenido de https://riptutorial.com/es/home: https://riptutorial.com/Download/typescript-es.pdf

Copes, F. (12 de Marzo de 2018). *Keep your code clean with ESLint*. Obtenido de https://flaviocopes.com:

https://flaviocopes.com/eslint/

Domínguez, C. M. (27 de Junio de 2019). *Aplicación web para la visualización de eventos*. Obtenido de https://riull.ull.es/xmlui/bitstream/handle/915/15482/Aplicacion%20web%20para%20la%20visualizacion%20de%20eventos..pdf?sequence=1&isAllowed=y

ESLINT. (20 de enero de 2014). Obtenido de https://eslint.org/

Foundation, L. (10 de diciembre de 2018). *Acerca de Node.js*. Obtenido de https://nodejs.org: https://nodejs.org/es/about/

Galiano, F. B. (08 de marzo de 2018). *Las etapas del proceso de desarrollo de software*. Obtenido de https://elvex.ugr.es/: https://elvex.ugr.es/idbis/db/docs/lifecycle.pdf

Gonzalez, P. (22 de Junio de 2019). *TypeScript ESLint*. Obtenido de TypeScript ESLint: https://github.com/typescript-eslint/typescript-eslint#what-are-eslint-and-typescript-and-how-do-they-compare

Hoyos, S. (16 de Marzo de 2018). *Manten Limpio tu código JavaScript usando Linters, EditorConfig y Prettier*. Obtenido de https://medium.com/@simonhoyos/manten-limpio-tu-c%C3%B3digo-javascript-usando-linters-editorconfig-y-prettier-25dadc638b99

Kraffczyk, J. F. (14 de Mayo de 2003). *Ingeniería de software, análisis y diseño*. Obtenido de http://catarina.udlap.mx/: http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/fuentes_k_jf/capitulo2.pdf

Martín, J. R. (18 de Marzo de 2019). *ESLint*. Obtenido de ESLint: https://lenguajejs.com/p/javascript/caracteristicas/eslint

Pozzi, E. F. (14 de 10 de 2017). *Que es un Linter y como podemos utilizarlo en nuestro entorno de desarrollo*. Obtenido de https://desarrolloactivo.com: https://desarrolloactivo.com/articulos/linter-standard-babel-eslint/

Rodriguez, G. (05 de Julio de 2017). *Metodología para el analisis y diseño de sistemas*. Obtenido de https://es.slideshare.net/: https://es.slideshare.net/GermanRodriguez93/metodologias-para-el-analisis-y-diseo-de-sistemas-77553977

Tómasdóttir, K., Aniche, M., & van Deursen, A. (2018). *https://www.semanticscholar.org/search?q=eslint&sort=relevance*. Obtenido de The Adoption of JavaScript Linters in Practice: A Case Study on ESLint: https://pdfs.semanticscholar.org/2c96/4d9d0a6b02d76f536c40e6c1ac507c575f81.pdf?_ga=2.211064277.1996440540.1571365045-300205488.1571365045

Williams, B. (23 de Marzo de 2019). *Testing Your Style With ESLint and Mocha*. Obtenido de https://thoughtbot.com: https://thoughtbot.com/blog/testing-your-style-with-eslint-and-mocha